동국대학교
dongguk university

형식언어

Mini C Scanner 만들기 프로젝트

2014112025 권민준

제출일 2016. 05. 17.

1. 문제분석

목표

ANSI C의 완전한 Subset인 **Mini C의 Scanner를 객체지향언어로 프로그램 작성**

> Mini C는 기존의 ANSI C 문법으로부터 몇 가지 언어 구조를 축소하여 고안하였다. 전체적인 프로그램 구조는 유사하지만 자료형은 정수형만 있고 다중 배열과 같은 구조는 존재하지 않는다. 또한, 비트 관련 연산자는 제외하였다. 그러나, 일반적으로 실험용 컴파일러를 설계해 보는 데는 교육적으로 큰 효과가 있다고 생각된다.

프로젝트 명세

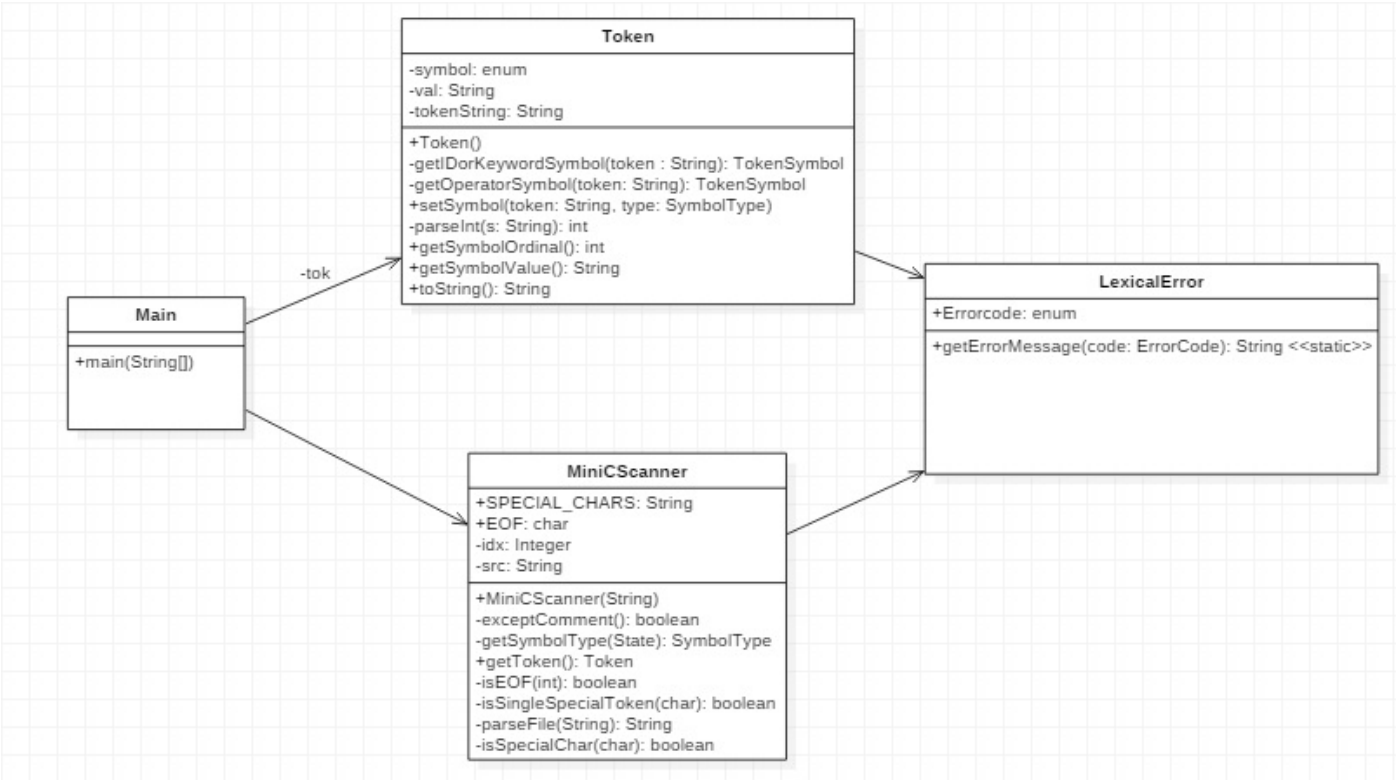
Mini C 소스코드(*.mc) 를 입력받아 **Token을 추출해내는 프로그램**

- **주석문(Comment)** 의 종류 - 2가지
  - `Block Comment` (/* ... */)
  - `Line Comment` (// ...)
- **명칭(Identifier)의 형태**
  - [a-zA-Z_][a-zA-Z0-9_]*
  - 예약어(Reserved Word)는 명칭으로 사용할 수 없음.
- **정수 상수 (Integer Constant)**
  - **10진수(Decimal Number)** : 0으로 시작하지 않는 일반 숫자. (eg. 526)
  - **8진수(Octal Number)** : 0으로 시작하는 숫자. (eg. 0526)
  - **16진수(Hexa-decimal Number)** : 0x로 시작하는 숫자 혹은 알파벳 A~F.(1~F) (eg. 0xFF, 0xff)
- **지정어(Keyword)**
  - `const`, `else`, `if`, `int`, `return`, `void`, `while`
- **연산자(Operator)**
  - 사칙 연산자 : **+**, **-**, *, **/**, **%**
  - 배정 연산자 : **=**, **+=**, **-=**, ***=**, **/=**, **%=**
  - 논리 연산자 : **!**, **&&**, **||**
  - 관계 연산자 : **==**, **!=**, **<**, **>**, **<=**, **>=**
  - 증감 연산자 : **++**, **--**
- 이외, **대괄호('[', ']')**, **중괄호('{', '}')**, **소괄호('(', ')')**, **컴마(',')**, **세미콜론(';')**.

2. 프로그램 설계, 알고리즘 설명

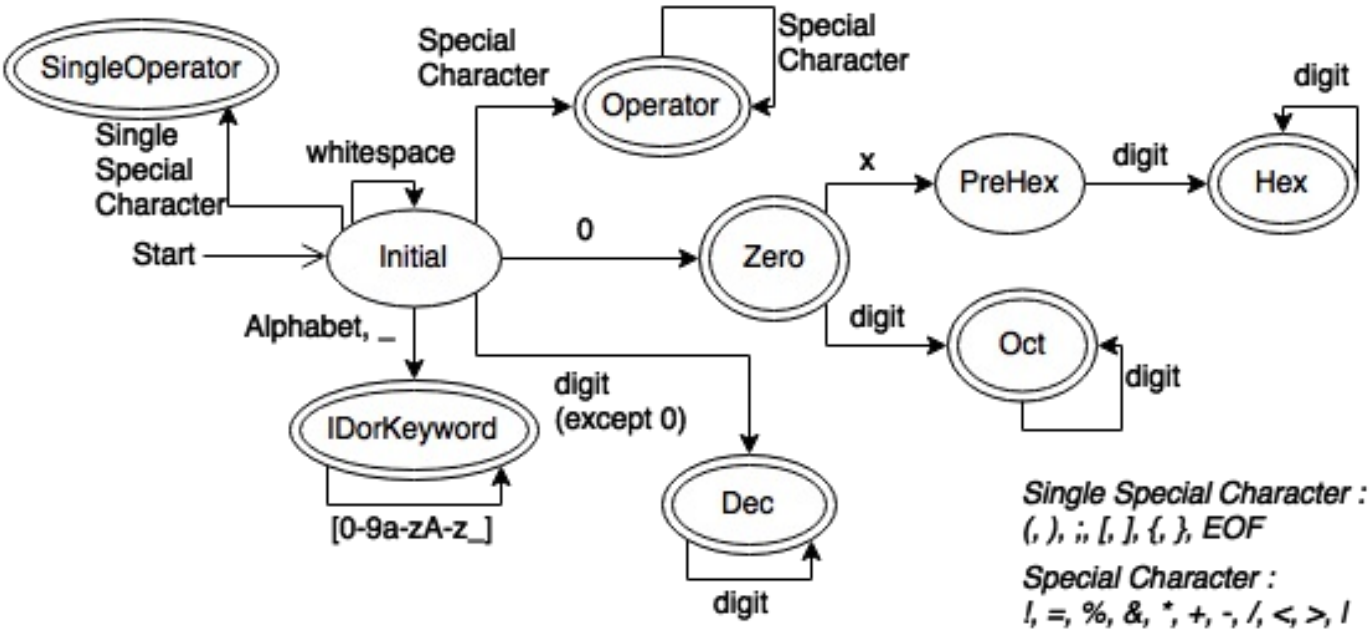
객체지향언어인 **Java** 를 이용해 프로그램을 작성했습니다.

클래스 다이어그램(Class Diagram)



프로그램을 실행할 때, 디버그 인자로 **소스코드 파일 경로** 를 입력받아 `MiniCScanner` 객체를 생성하면서 전달합니다.
`MiniCScanner` 객체에서 소스코드 파일 경로를 입력받으면 `BufferedReader` 객체를 이용해 소스코드를 전부 읽어들여 String 타입 `멤버변수 src` 에 저장합니다. 그 이후에 Main 함수에서 MiniCScanner 객체의 `getToken()` 메소드를 호출하면 읽어들인 소스코드에서 순차적으로 Token을 추출해냅니다.
`getToken()` 메소드는 소스코드를 읽으면서 **유효한 토큰 String** 을 잘라냅니다. String을 잘라낼 때, 해당 토큰이 어떤 심볼을 의미하는 지 크게 나누기 위해 **토큰 심볼의 타입** 을 크게 3개(`Operator` , `IDorKeyword` , `Digit`)로 분류합니다. 이러한 심볼의 타입들을 구분하기 위해 Scanner에서는 **9개의 State** 를 사용했습니다. 그 State들을 나타내는 **State Diagram** 은 **다음 페이지에 첨부했습니다.**

상태 전이도(State Diagram)



State를 통해 인식된 Token들은 `Token` 객체를 생성하여 Token에 해당하는 **Symbol** 을 설정합니다. `Token` 객체에서는 MiniCScanner 객체에서 전달받은 **토큰 String과 심볼의 대분류 타입** 을 전달받아 정확한 Symbol을 설정합니다. 토큰이 완성되고 나면 MiniCScanner 객체는 완성된Token 객체를 반환합니다.

추가적으로, Error를 처리하기 위한 `LexicalError` 클래스를 만들었습니다. 에러의 종류를 나누는 `ErrorCode` Enumerator를 생성하고, 그에 해당하는 String을 만들어 출력합니다.

3. 주석이 첨부된 중요 소스코드 및 설명

MiniCScanner.java

토큰을 추출해내는 Core 클래스

- **getToken()** : 소스코드를 순차적으로 읽어 하나의 토큰을 추출합니다.
- **parseFile()** : 소스코드 파일 경로를 통해 파일의 내용을 읽어 String으로 변환합니다.
- **isEOF()** : 현재 Scanner가 읽고있는 커서가 파일의 끝인지 확인합니다.
- **isSpecialChar()** : 문자 하나를 입력받아 특수문자 *(해당 글자와 다른글자의 조합으로 토큰이 될 수 있는 경우 제외)* 인지 검사합니다.
- **isSingleSpecialToken()** : 문자 하나를 입력받아 1글자 자체 *(해당 글자와 다른글자의 조합으로 토큰이 될 수 있는 경우)* 가 토큰인지 검사합니다.
- **getSymbolType()** : State를 입력받아 인식되는 토큰의 심볼을 반환합니다. (대분류)
- **exceptComment()** : 현재 Scanner가 읽고있는 커서로부터 유효한 토큰이 나오기 전까지의 주석들을 모두 제거합니다.

```
package com.minjunkweon;
```

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

/**
 * Token을 가져오기 위한 처리를 담당하는 Scanner 클래스
 * Created by kweonminjun on 2016. 5. 13..
 */
public class MiniCScanner {
    static public final char EOF = '\255'; // 파일의 끝을 의미하는 EOF 문자 상수
    static public final String SPECIAL_CHARS = "!=%&*+-/<>|"; // 두 글자 이상이 하나의 토큰일 수 있는 특수문자들
    static public final int ID_LENGTH = 12; // 컴파일러를 구현할 때에는 명칭의 길이에 제한을 두는 것이 좋다

    private String src; // Source Code의 전체 내용을 String으로 저장하기 위한 변수
    private Integer idx; // Source Code를 읽을 때 cursor 역할을 하는 변수

    /**
     * Token을 추출해낼 때 어떤 토큰을 인식하고 있는지 나타내기 위한 State
     */
    private enum State {
        Initial, Dec, Oct, Hex, IDorKeyword, Operator, Zero, PreHex, SingleOperator
    }

    /**
     * Mini C Scanner 생성자
     * 소스 코드의 파일 경로를 입력받아 src 변수에 String으로 저장하고, 커서를 맨 처음으로 이동
     *
     * @param filePath - 소스 코드의 파일 경로
     */
    public MiniCScanner(String filePath) {
        src = parseFile(filePath);
        idx = 0;
    }

    /**
     * 소스코드 경로를 통해 소스코드 파일을 String으로 읽어 들이는 Method
     *
     * @param filePath - 읽어올 소스코드 경로
     * @return 소스코드 파일의 내용 (String)
     */
    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 경로
를 이용해 File Reader 생성
        } catch (IOException e) {
            // 파일을 읽을 수 없음

System.err.print(LexicalError.getErrorMessage(LexicalError.ErrorCode.C
annotOpenFile));
            return "";
        }

        BufferedReader reader = new BufferedReader(fileReader); //
BufferedReader 객체를 만들어 소스코드 파일을 읽음
        try {
            while ((readedString = reader.readLine()) != null) // 파일로
부터 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(LexicalError.getErrorMessage(LexicalError.ErrorCode.C
annotOpenFile));
            return "";
        }
        return src;
    }

    /**
     * Core Method
     * Source code에서 Token 단위로 String을 나누고 Token 객체를 만들어 반환하는
처리를 한다
     *
     * @return 인식된 토큰 객체 반환
     */
    public Token getToken() {
        Token token = new Token();
        Token.SymbolType symType = Token.SymbolType.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(LexicalError.getErrorMessage(LexicalError.ErrorCode.I
nvalidComment));
            return token;
        }

        while (!isEOF(idx)) { // 소스코드를 전부 읽을때 까지 계속
            char c = src.charAt(idx++); // 커서로부터 글자 하나를 읽고 커서를
한칸 이동

            if (Character.isWhitespace(c)) { // white space (needs
trimming)
                if(state != State.Initial) break; // 만약 글자들을 인식하고
있었다면 그대로 종결
                else continue;
            } else if (isSingleSpecialToken(c)) { // signle operator (
'(', ')', '{', '}', ',', '[', ']', ';', EOF )
                if (state == State.Initial) { // 처음부터 1글자짜리
                    state = State.SingleOperator;
                    tokenString = String.valueOf(c);
                } else { --idx; }
                break;
            } else if (isSpecialChar(c)) { // 1글자짜리 연산자가 아닌 2글자 이
상의 연산자가 될 수 있는 연산자일 경우
                if (state != State.Initial && state != State.Operator)
{
                    --idx; break;
                }
                state = State.Operator;
            } else if (state == State.Initial && c == '0') { // Zero를
인식할 경우
                state = State.Zero;
            } else if (Character.isDigit(c)) { // 숫자를 인식한 경우
                if (state == State.Initial) // 아무것도 인식하지 않았을 경우,
10진수로 취급
                    state = State.Dec;
                else if (state == State.Zero) // 숫자 0을 인식하고 있었을 경
우, 8진수로 취급
                    state = State.Oct;
                else if (state == State.PreHex) // 0x 까지 인식하고 있었을
경우, 16진수로 취급
                    state = State.Hex;
                else if (state == State.Operator) { // 연산자가 나온 뒤 숫
자가 나올경우 while문 탈출
                    --idx; break;
                }
            } else if (state == State.Zero && c == 'x') { // 0x까지 인식
했을 경우
                state = State.PreHex;
            } else if (Character.isAlphabetic(c) || c == '_') { //
```

```
underscore 혹은 알파벳을 인식했을 경우
                if (state != State.Initial && state !=
State.IDorKeyword) {
                    // 명칭 혹은 키워드가 아닌 토큰을 인식하는 중일 경우 while문 탈
출
                    --idx; break;
                }
                state = State.IDorKeyword; // 명칭 혹은 키워드 인식
            }

            tokenString += String.valueOf(c); // 토큰 String에 글자 추가
        }
        symType = getSymbolType(state); // 인식한 state로부터 토큰이 어떤 값을
의미하는지 대분류
        if (symType == Token.SymbolType.IDorKeyword &&
tokenString.length() > ID_LENGTH) {
            // 명칭의 길이가 제한을 넘어갈 경우 에러로 처리
            // ERROR : 명칭의 길이 초과

System.err.print(LexicalError.getErrorMessage(LexicalError.ErrorCode.A
boveIDLimit));
            return token;
        }
        token.setSymbol(tokenString, symType); // tokenString과 함께 대분
류한 타입을 전달하여 token을 세팅
        return token; // 인식한 token을 반환
    }

    /**
     * 커서가 파일의 끝을 가리키고 있는지 확인하는 Method
     * 즉, 더이상 읽을 문자가 없는지 확인하는 Method
     *
     * @param idx - 확인할 위치
     * @return 더이상 읽을 문자가 없으면 true, 아니면 false
     */
    private boolean isEOF(int idx) {
        return idx >= src.length();
    }

    /**
     * 문자가 특수문자(1글자 연산자 제외)인지 확인하는 Method
     *
     * @param c - 확인의 대상 문자
     * @return 특수문자(1글자 연산자 제외)일 경우 true 반환, 아닐경우 false 반환
     */
    private boolean isSpecialChar(char c) {
        for (int i = 0; i < SPECIAL_CHARS.length(); ++i)
            if (SPECIAL_CHARS.charAt(i) == c)
                return true;
```

```
        return false;
    }

    /**
     * 문자가 1글자 연산자인지 확인하는 Method
     *
     * @param c - 확인의 대상 문자
     * @return 1글자 연산자일 경우 true 반환, 아닐경우 false 반환
     */
    private boolean isSingleSpecialToken(char c) {
        switch (c) {
            case '(': case ')': case ',': case ';':
            case '[': case ']': case '{': case '}':
            case EOF:
                return true;
            default:
                return false;
        }
    }

    /**
     * 추출한 token의 타입이 어떤 타입인지 state에 따라 분류
     *
     * @param s - 인식된 state
     * @return state에 해당하는 심볼 타입
     */
    private Token.SymbolType getSymbolType(State s) {
        switch (s) {
            // 숫자일 경우 (10진수, 8진수, 16진수, 0)
            case Dec:
            case Oct:
            case Hex:
            case Zero:
                return Token.SymbolType.Digit;
            // 명칭 혹은 키워드일 경우
            case IDorKeyword:
                return Token.SymbolType.IDorKeyword;
            // 연산자일 경우
            case Operator:
            case SingleOperator:
                return Token.SymbolType.Operator;
            // 종결상태가 아닌 State의 경우 NULL Type을 반환 (인식실패)
            case Initial:
            case PreHex:
            default:
                return Token.SymbolType.NULL;
        }
    }

    /**
```

```
     * 현재 소스코드 파일에 대한 커서(idx)로부터 유효한 토큰이 나올 때까지 주석을 무시하는 Method
     *
     * @return block comment에 에러가 있을 경우 true, 아니면 false 반환
     */
    private boolean exceptComment() {
        char c;
        // 커서로부터 whitespace 문자들 모두 무시
        while (!isEOF(idx) && Character.isWhitespace(src.charAt(idx))) idx++;
        if (isEOF(idx)) return false; // whitespace 문자들을 모두 무시했는데 파일의 끝일 경우 성공적으로 주석 제거를 했다고 반환

        if (src.charAt(idx) == '/') { // '/'가 나올 경우
            if (src.charAt(idx+1) == '/') {    // Line Comment
                idx += 2; // "//" 다음 문자로 커서 이동
                while (!isEOF(idx) && src.charAt(idx) != '\n') idx++; // 개행문자 혹은 EOF 문자가 나올 때까지 커서 이동
                if (!isEOF(idx)) idx += 1; // 개행문자 다음에 문자가 있을 경우, 그 문자로 커서 이동
            } else if (src.charAt(idx+1) == '*') { // Block Comment
                idx += 2; // "/*" 다음 문자로 커서 이동
                while (src.charAt(idx) != '*' && src.charAt(idx+1) != '/') { // */가 나올때 까지
                    if (isEOF(idx+1)) return true; // 불완전한 block comment이므로 에러가 있다고 반화
                    idx++;
                }
                idx += 2; // "*/" 다음 문자로 커서 이동
            }
        }
        return false; // 에러가 없이 성공적으로 주석 제거
    }
}
```

LexicalError.java

에러들의 종류를 분류하고 출력하기 위한 클래스

- **getErrorMessage()** : 에러 코드에 해당되는 에러 메시지를 출력합니다.

```
package com.minjunkweon;

/**
 * Lexical Error를 처리하는 클래스
 * Created by kweonminjun on 2016. 5. 13..
 */
public class LexicalError {
```

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

    public static String getErrorMessage(ErrorCode code) {
        String msg;
        msg = "Lexical Error(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;
    }
}
```

Token.java

Scanner에서 인식된 토큰 String을 입력받아 정확한 Symbol을 배정하고 객체 자체가 Token이 되는 모델 클래스

- **getIDorKeywordSymbol()** : 대분류가 `IDorKeyword`(명칭 혹은 지정어)인 경우 정확히 어떤 심볼을 갖는지 찾습니다.
- **getOperatorSymbol()** : 대분류가 `Operator`(연산자)인 경우 정확히 어떤 심볼을 갖는지 찾습니다.
- **setSymbol()** : Token의 Symbol과 value를 설정하는 메소드. `MiniCScanner` 객체에서 잘라낸 토큰 String에 맞는 Symbol로 설정합니다.

- **parseInt()** : 16진수, 8진수, 10진수에 상관없이 String을 정수형으로 추출해냅니다. (eg. 0x1F, 047, 14)
- **getSymbolOrdinal()** : Symbol에 해당하는 토큰 심볼을 숫자로 표현합니다.
- **getSymbolValue()** : 토큰이 명칭(Identifier)이나 정수(Number)일 경우, 그 토큰 값을 얻습니다.
- **toString()** : 출력을 편하게 하기 위한 toString 메소드. System.out.println()에 객체를 넣으면 이 메소드가 묵시적으로 실행됩니다.

```
package com.minjunkweon;

/**
 * Mini C Scanner에서 사용할 Token의 Model 클래스
 * Created by kweonminjun on 2016. 5. 13..
 */
public class Token {
    /**
     * Mini C Scanner 객체에서 인식한 Token의 Symbol 타입 (대분류)
     */
    public enum SymbolType {
        Operator, IDorKeyword, Digit, NULL
    }

    /**
     * Token의 키워드나 명칭, 연산자를 식별하기 위한 Symbol (소분류)
     */
    public enum TokenSymbol {
        NULL,
        Not, NotEqu, Mod, ModAssign, ID, Number,
        And, LParen, RParen, Mul, MulAssign, Plus,
        Increase, AddAssign, Comma, Minus, Decrease, SubAssign,
        Div, DivAssign, Semicolon, Less, Lesser, Assign,
        Equal, Great, Greater, LBracket, RBracket, EOF,
        Const, Else, If, Int, Return, Void, While,
        LBrace, Or, RBrace
    }

    private TokenSymbol symbol; // Token이 가진 Symbol (Symbol에 상응하는 정수를 추출해낼 수 있다)
    private String val; // 명칭 혹은 숫자일 경우 그 값을 저장
    private String tokenString; // 인식한 토큰의 원시 String

    /**
     * 생성자
     * 각각의 멤버변수들을 NULL로 초기화한다.
     */
    public Token() {
        symbol = TokenSymbol.NULL;
        val = "0";
        tokenString = "NULL";
```

```
    }

    /**
     * 키워드나 명칭을 입력받았을 때, 그 String이 키워드인지 명칭인지 구분하는 메소드
     * 키워드라면 어떤 키워드인지 Symbol을 할당
     *
     * @param token - Symbol을 구분할 토큰 String
     * @return 구분된 토큰 Symbol
     */
    private TokenSymbol getIDorKeywordSymbol(String token) {
        switch (token) {
            // Keyword
            case "const":   return TokenSymbol.Const;
            case "else":    return TokenSymbol.Else;
            case "if":      return TokenSymbol.If;
            case "int":     return TokenSymbol.Int;
            case "return":  return TokenSymbol.Return;
            case "void":    return TokenSymbol.Void;
            case "while":   return TokenSymbol.While;

            // ID
            default:
                return TokenSymbol.ID;
        }
    }

    /**
     * 입력받은 토큰 String이 연산자라면 어떤 연산자인지 구분하기 위한 메소드
     *
     * @param token - Symbol을 구분할 토큰 String
     * @return 구분된 토큰 Symbol (연산자)
     */
    private TokenSymbol getOperatorSymbol(String token) {
        switch (token) {
            case "!":   return TokenSymbol.Not;
            case "!=":  return TokenSymbol.NotEqu;
            case "%":   return TokenSymbol.Mod;
            case "%=":  return TokenSymbol.ModAssign;
            case "&&":  return TokenSymbol.And;
            case "(":   return TokenSymbol.LParen;
            case ")":   return TokenSymbol.RParen;
            case "*":   return TokenSymbol.Mul;
            case "*=":  return TokenSymbol.MulAssign;
            case "+":   return TokenSymbol.Plus;
            case "++":  return TokenSymbol.Increase;
            case "+=":  return TokenSymbol.AddAssign;
            case ",":   return TokenSymbol.Comma;
            case "-":   return TokenSymbol.Minus;
            case "--":  return TokenSymbol.Decrease;
            case "-=":  return TokenSymbol.SubAssign;
```

```
            case "/":   return TokenSymbol.Div;
            case "/=":  return TokenSymbol.DivAssign;
            case ";":   return TokenSymbol.Semicolon;
            case "<":   return TokenSymbol.Less;
            case "<=":  return TokenSymbol.Lesser;
            case "=":   return TokenSymbol.Assign;
            case "==":  return TokenSymbol.Equal;
            case ">":   return TokenSymbol.Great;
            case ">=":  return TokenSymbol.Greater;
            case "[":   return TokenSymbol.LBracket;
            case "]":   return TokenSymbol.RBracket;
            case "\255": return TokenSymbol.EOF;
            case "{":   return TokenSymbol.LBrace;
            case "||":  return TokenSymbol.Or;
            case "}":   return TokenSymbol.RBrace;
            case "&":

System.err.print(LexicalError.getErrorMessage(LexicalError.ErrorCode.S
ingleAmpersand));
                break;
            case "|":

System.err.print(LexicalError.getErrorMessage(LexicalError.ErrorCode.S
ingleBar));
                break;
            default: // 인식하지 못한 TokenSymbol

System.err.print(LexicalError.getErrorMessage(LexicalError.ErrorCode.I
nvalidChar));
                break;
        }
        return TokenSymbol.NULL;
    }

    /**
     * Token의 Symbol과 value를 설정하는 메소드
     * Mini C Scanner 객체에서 잘라낸 토큰 String을 통해 그 String에 맞는
Symbol로 설정한다
     *
     * @param token - 잘라낸 토큰 String
     * @param type - Mini C Scanner 객체에서 분류한 타입 (키워드나 명칭, 숫자,
연산자로 대분류)
     */
    public void setSymbol(String token, SymbolType type) {
        tokenString = token;
        switch (type) {
            case IDorKeyword:
                symbol = getIDorKeywordSymbol(token);
                if (symbol == TokenSymbol.ID) // 명칭일 경우
                    val = token;
```

```
                break;
            case Digit:
                symbol = TokenSymbol.Number;
                val = Integer.toString(parseInt(token));
                break;
            case Operator:
                symbol = getOperatorSymbol(token);
                break;
            case NULL:
            default:
                break;
        }
    }

    /**
     * 16진수, 8진수, 10진수에 상관없이 String을 정수형으로 추출해내는 메소드
     *
     * @param s - 정수로 변환할 String (eg. 0x1F, 047, 14)
     * @return String의 정수
     */
    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
    }

    /**
     * Symbol에 해당하는 토큰 심볼을 숫자로 표현하는 메소드
     *
     * @return 토큰 심볼의 숫자 (-1 : NULL)
     */
    public int getSymbolOrdinal() {
        return symbol.ordinal()-1;  // NULL이 -1이기 때문에 -1 해야한다.
    }

    /**
     * 토큰이 명칭이나 숫자일 경우 그 토큰 값을 얻는 메소드
     *
     * @return 토큰 값 (없을 경우 0 반환)
     */
    public String getSymbolValue() {
        return val;
    }

    /**
```

```
     * 출력하기 편하게 하기 위해 정의하는 toString 메소드
     *
     * @return 토큰을 표현하기 위한 String
     */
    public String toString() {
        return tokenString + "\t : (" + getSymbolOrdinal() + ", "+
getSymbolValue() + ")";
    }
}
```

Main.java

프로그램을 실행하기 위한 main() 메소드가 포함된 클래스

- **main()** : Entry Point 메소드. 프로그램의 시작되는 메소드가 됩니다. *(C언어의 main()과 같은 역할)*

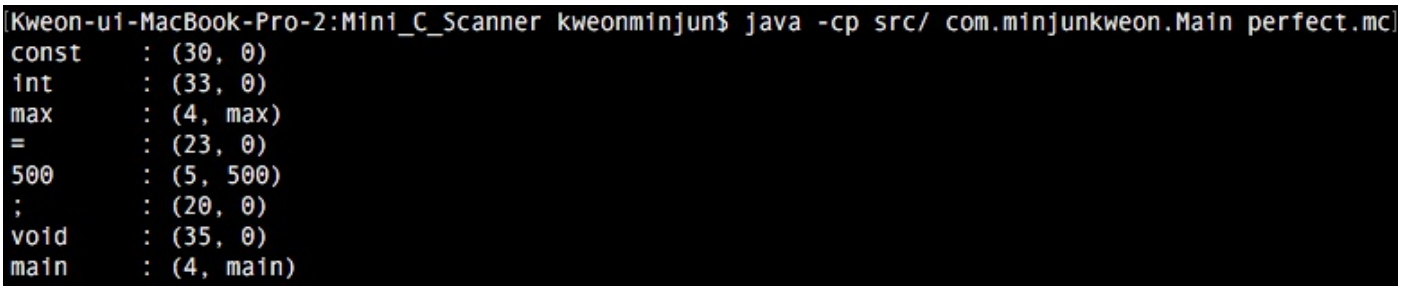
```
package com.minjunkweon;

/**
 * 프로그램의 흐름을 담당하는 Main 클래스
 * Created by kweonminjun on 2016. 5. 13..
 */
public class Main {
    /**
     * Entry Point 메소드
     *
     * @param args - 디버그 인자 (0번째 원소로 입력 파일 경로를 받음)
     */
    public static void main(String[] args) {
        if (args[0] == null) { // 스캐너가 분석할 파일의 경로를 받지 못했을 경우
에러 출력
            System.err.print("Please enter the file path (by debug
argument)");
            return;
        }
        MiniCScanner sc = new MiniCScanner(args[0]); // Mini C Scanner
객체
        Token tok = null; // Mini C Scanner에서 얻어낸 token을 저장하기 위한
Token 변수
        while ((tok = sc.getToken()).getSymbolOrdinal() != -1) // Mini
C Scanner에서 다 읽을 때까지 token을 얻어 출력
            System.out.println(tok);
    }
}
```

4. 결과화면 및 결과분석

컴파일러 입문 교재 부록으로 첨부되어 있는 Mini C의 예제 소스코드를 프로그램의 입력으로 넣었을 때, 출력결과들입니다. 출력결과가 너무 많아서 스크린샷은 일부만 캡처하였고, 전체 내용은 아래에 첨부했습니다.
출력형식은 **<토큰> : <심볼 넘버> <토큰 값>** 이며, 토큰 값은 토큰이 **명칭(Identifier)이거나 정수(Number)일 경우** 에만 존재합니다.

perfect.mc 파일 프로그램 실행결과

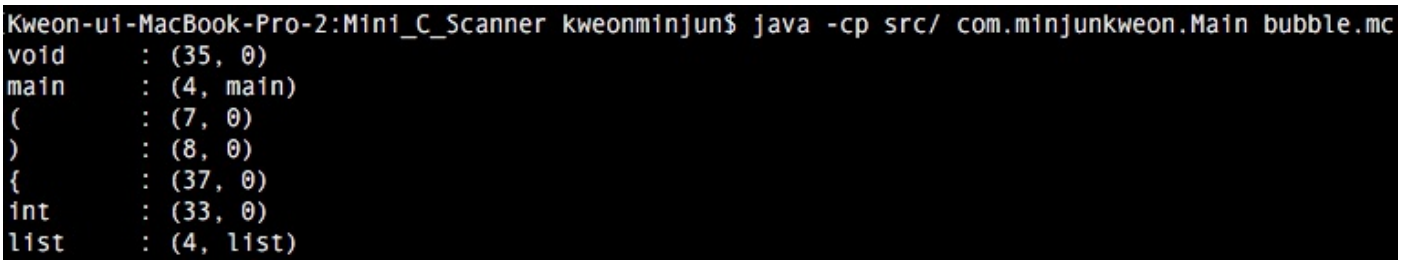



```
const    : (30, 0)
int      : (33, 0)
max      : (4, max)
=        : (23, 0)
500      : (5, 500)
;        : (20, 0)
void     : (35, 0)
main     : (4, main)
(        : (7, 0)
)        : (8, 0)
{        : (37, 0)
int      : (33, 0)
i        : (4, i)
,        : (14, 0)
j        : (4, j)
,        : (14, 0)
k        : (4, k)
;        : (20, 0)
int      : (33, 0)
rem      : (4, rem)
,        : (14, 0)
sum      : (4, sum)
;        : (20, 0)
i        : (4, i)
=        : (23, 0)
2        : (5, 2)
;        : (20, 0)
while    : (36, 0)
(        : (7, 0)
```

```
i         : (4, i)
<=        : (22, 0)
max       : (4, max)
)         : (8, 0)
{         : (37, 0)
sum       : (4, sum)
=         : (23, 0)
0         : (5, 0)
;         : (20, 0)
k         : (4, k)
=         : (23, 0)
i         : (4, i)
/         : (18, 0)
2         : (5, 2)
;         : (20, 0)
j         : (4, j)
=         : (23, 0)
1         : (5, 1)
;         : (20, 0)
while     : (36, 0)
(         : (7, 0)
j         : (4, j)
<=        : (22, 0)
k         : (4, k)
)         : (8, 0)
{         : (37, 0)
rem       : (4, rem)
=         : (23, 0)
i         : (4, i)
%         : (2, 0)
j         : (4, j)
;         : (20, 0)
if        : (32, 0)
(         : (7, 0)
rem       : (4, rem)
==        : (24, 0)
0         : (5, 0)
)         : (8, 0)
sum       : (4, sum)
+=        : (13, 0)
j         : (4, j)
;         : (20, 0)
++        : (12, 0)
j         : (4, j)
;         : (20, 0)
}         : (39, 0)
if        : (32, 0)
(         : (7, 0)
i         : (4, i)
==        : (24, 0)
```

```
sum       : (4, sum)
)         : (8, 0)
write     : (4, write)
(         : (7, 0)
i         : (4, i)
)         : (8, 0)
;         : (20, 0)
++        : (12, 0)
i         : (4, i)
;         : (20, 0)
}         : (39, 0)
}         : (39, 0)
          : (29, 0)
```

bubble.mc 파일 프로그램 실행결과




```
void      : (35, 0)
main      : (4, main)
(         : (7, 0)
)         : (8, 0)
{         : (37, 0)
int       : (33, 0)
list      : (4, list)
[         : (27, 0)
100       : (5, 100)
]         : (28, 0)
;         : (20, 0)
int       : (33, 0)
element   : (4, element)
;         : (20, 0)
int       : (33, 0)
total     : (4, total)
,         : (14, 0)
i         : (4, i)
,         : (14, 0)
top       : (4, top)
;         : (20, 0)
int       : (33, 0)
temp      : (4, temp)
```

```
;         : (20, 0)
i         : (4, i)
=         : (23, 0)
1         : (5, 1)
;         : (20, 0)
read      : (4, read)
(         : (7, 0)
element   : (4, element)
)         : (8, 0)
;         : (20, 0)
while     : (36, 0)
(         : (7, 0)
element   : (4, element)
!=        : (1, 0)
0         : (5, 0)
)         : (8, 0)
{         : (37, 0)
list      : (4, list)
[         : (27, 0)
i         : (4, i)
]         : (28, 0)
=         : (23, 0)
element   : (4, element)
;         : (20, 0)
++        : (12, 0)
i         : (4, i)
;         : (20, 0)
read      : (4, read)
(         : (7, 0)
element   : (4, element)
)         : (8, 0)
;         : (20, 0)
}         : (39, 0)
top       : (4, top)
=         : (23, 0)
total     : (4, total)
=         : (23, 0)
i         : (4, i)
-         : (15, 0)
1         : (5, 1)
;         : (20, 0)
while     : (36, 0)
(         : (7, 0)
top       : (4, top)
>         : (25, 0)
1         : (5, 1)
)         : (8, 0)
{         : (37, 0)
i         : (4, i)
=         : (23, 0)
```

```
1         : (5, 1)
;         : (20, 0)
while     : (36, 0)
(         : (7, 0)
i         : (4, i)
<         : (21, 0)
top       : (4, top)
)         : (8, 0)
{         : (37, 0)
if        : (32, 0)
(         : (7, 0)
list      : (4, list)
[         : (27, 0)
i         : (4, i)
]         : (28, 0)
>         : (25, 0)
list      : (4, list)
[         : (27, 0)
i         : (4, i)
+         : (11, 0)
1         : (5, 1)
]         : (28, 0)
)         : (8, 0)
{         : (37, 0)
temp      : (4, temp)
=         : (23, 0)
list      : (4, list)
[         : (27, 0)
i         : (4, i)
]         : (28, 0)
;         : (20, 0)
list      : (4, list)
[         : (27, 0)
i         : (4, i)
]         : (28, 0)
=         : (23, 0)
list      : (4, list)
[         : (27, 0)
i         : (4, i)
+         : (11, 0)
1         : (5, 1)
]         : (28, 0)
;         : (20, 0)
list      : (4, list)
[         : (27, 0)
i         : (4, i)
+         : (11, 0)
1         : (5, 1)
]         : (28, 0)
=         : (23, 0)
```

```
temp      : (4, temp)
;         : (20, 0)
}         : (39, 0)
++        : (12, 0)
i         : (4, i)
;         : (20, 0)
}         : (39, 0)
top       : (4, top)
--        : (16, 0)
;         : (20, 0)
}         : (39, 0)
i         : (4, i)
=         : (23, 0)
1         : (5, 1)
;         : (20, 0)
while     : (36, 0)
(         : (7, 0)
i         : (4, i)
<=        : (22, 0)
total     : (4, total)
)         : (8, 0)
{         : (37, 0)
write     : (4, write)
(         : (7, 0)
list      : (4, list)
[         : (27, 0)
i         : (4, i)
]         : (28, 0)
)         : (8, 0)
;         : (20, 0)
++        : (12, 0)
i         : (4, i)
;         : (20, 0)
}         : (39, 0)
}         : (39, 0)
          : (29, 0)
```

factorial.mc 파일 프로그램 실행결과

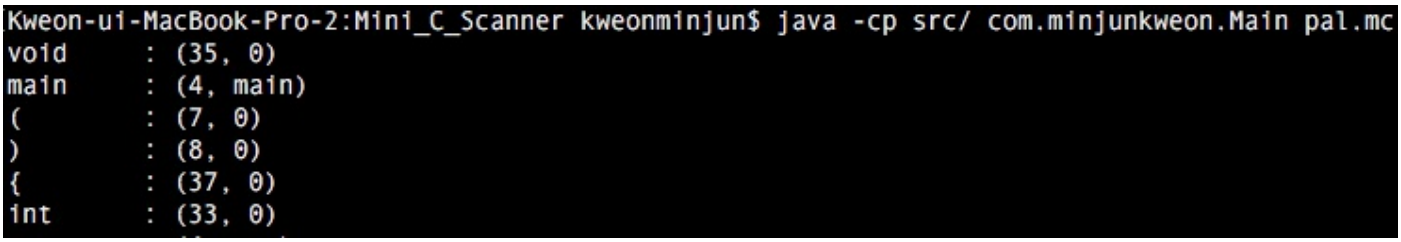
```
Kweon-ui-MacBook-Pro-2:Mini_C_Scanner kweonminjun$ java -cp src/ com.minjunkweon.Main factorial.mc
void      : (35, 0)
main      : (4, main)
(         : (7, 0)
)         : (8, 0)
{         : (37, 0)
int       : (33, 0)
```

```
void      : (35, 0)
```

```
main     : (4, main)
(        : (7, 0)
)        : (8, 0)
{        : (37, 0)
int      : (33, 0)
n        : (4, n)
,        : (14, 0)
f        : (4, f)
;        : (20, 0)
read     : (4, read)
(        : (7, 0)
n        : (4, n)
)        : (8, 0)
;        : (20, 0)
write    : (4, write)
(        : (7, 0)
n        : (4, n)
)        : (8, 0)
;        : (20, 0)
f        : (4, f)
=        : (23, 0)
factorial        : (4, factorial)
(        : (7, 0)
n        : (4, n)
)        : (8, 0)
;        : (20, 0)
write    : (4, write)
(        : (7, 0)
f        : (4, f)
)        : (8, 0)
;        : (20, 0)
}        : (39, 0)
int      : (33, 0)
factorial        : (4, factorial)
(        : (7, 0)
int      : (33, 0)
n        : (4, n)
)        : (8, 0)
{        : (37, 0)
if       : (32, 0)
(        : (7, 0)
n        : (4, n)
==       : (24, 0)
1        : (5, 1)
)        : (8, 0)
return   : (34, 0)
1        : (5, 1)
;        : (20, 0)
else     : (31, 0)
return   : (34, 0)
```

```
n         : (4, n)
*         : (9, 0)
factorial         : (4, factorial)
(         : (7, 0)
n         : (4, n)
-         : (15, 0)
1         : (5, 1)
)         : (8, 0)
;         : (20, 0)
}         : (39, 0)
          : (29, 0)
```

pal.mc 파일 프로그램 실행결과




```
void      : (35, 0)
main      : (4, main)
(         : (7, 0)
)         : (8, 0)
{         : (37, 0)
int       : (33, 0)
org       : (4, org)
,         : (14, 0)
rev       : (4, rev)
;         : (20, 0)
int       : (33, 0)
i         : (4, i)
,         : (14, 0)
j         : (4, j)
;         : (20, 0)
read      : (4, read)
(         : (7, 0)
org       : (4, org)
)         : (8, 0)
;         : (20, 0)
if        : (32, 0)
(         : (7, 0)
org       : (4, org)
<         : (21, 0)
0         : (5, 0)
```

```
)         : (8, 0)
org       : (4, org)
=         : (23, 0)
(         : (7, 0)
-         : (15, 0)
1         : (5, 1)
)         : (8, 0)
*         : (9, 0)
org       : (4, org)
;         : (20, 0)
i         : (4, i)
=         : (23, 0)
org       : (4, org)
;         : (20, 0)
rev       : (4, rev)
=         : (23, 0)
0         : (5, 0)
;         : (20, 0)
while     : (36, 0)
(         : (7, 0)
i         : (4, i)
!=        : (1, 0)
0         : (5, 0)
)         : (8, 0)
{         : (37, 0)
j         : (4, j)
=         : (23, 0)
i         : (4, i)
%         : (2, 0)
10        : (5, 10)
;         : (20, 0)
rev       : (4, rev)
=         : (23, 0)
rev       : (4, rev)
*         : (9, 0)
10        : (5, 10)
+         : (11, 0)
j         : (4, j)
;         : (20, 0)
i         : (4, i)
/=        : (19, 0)
10        : (5, 10)
;         : (20, 0)
}         : (39, 0)
if        : (32, 0)
(         : (7, 0)
rev       : (4, rev)
==        : (24, 0)
org       : (4, org)
)         : (8, 0)
```

```
write    : (4, write)
(        : (7, 0)
org      : (4, org)
)        : (8, 0)
;        : (20, 0)
}        : (39, 0)
         : (29, 0)
```

prime.mc 파일 프로그램 실행결과

```
Kweon-ui-MacBook-Pro-2:Mini_C_Scanner kweonminjun$ java -cp src/ com.minjunkweon.Main prime.mc
const    : (30, 0)
int      : (33, 0)
max      : (4, max)
=        : (23, 0)
100      : (5, 100)
;        : (20, 0)
void     : (35, 0)
```

```
const    : (30, 0)
int      : (33, 0)
max      : (4, max)
=        : (23, 0)
100      : (5, 100)
;        : (20, 0)
void     : (35, 0)
main     : (4, main)
(        : (7, 0)
)        : (8, 0)
{        : (37, 0)
int      : (33, 0)
i        : (4, i)
,        : (14, 0)
j        : (4, j)
,        : (14, 0)
k        : (4, k)
;        : (20, 0)
int      : (33, 0)
rem      : (4, rem)
,        : (14, 0)
prime    : (4, prime)
;        : (20, 0)
i        : (4, i)
=        : (23, 0)
2        : (5, 2)
;        : (20, 0)
while    : (36, 0)
(        : (7, 0)
```

```
i         : (4, i)
<=        : (22, 0)
max       : (4, max)
)         : (8, 0)
{         : (37, 0)
prime     : (4, prime)
=         : (23, 0)
1         : (5, 1)
;         : (20, 0)
k         : (4, k)
=         : (23, 0)
i         : (4, i)
/         : (18, 0)
2         : (5, 2)
;         : (20, 0)
j         : (4, j)
=         : (23, 0)
2         : (5, 2)
;         : (20, 0)
while     : (36, 0)
(         : (7, 0)
j         : (4, j)
<=        : (22, 0)
k         : (4, k)
)         : (8, 0)
{         : (37, 0)
rem       : (4, rem)
=         : (23, 0)
i         : (4, i)
%         : (2, 0)
j         : (4, j)
;         : (20, 0)
if        : (32, 0)
(         : (7, 0)
rem       : (4, rem)
==        : (24, 0)
0         : (5, 0)
)         : (8, 0)
prime     : (4, prime)
=         : (23, 0)
0         : (5, 0)
;         : (20, 0)
++        : (12, 0)
j         : (4, j)
;         : (20, 0)
}         : (39, 0)
if        : (32, 0)
(         : (7, 0)
prime     : (4, prime)
==        : (24, 0)
```

```
1         : (5, 1)
)         : (8, 0)
write     : (4, write)
(         : (7, 0)
i         : (4, i)
)         : (8, 0)
;         : (20, 0)
++        : (12, 0)
i         : (4, i)
;         : (20, 0)
}         : (39, 0)
}         : (39, 0)
          : (29, 0)
```

5. 소감

C언어로 작성된 절차지향적인 Lexical Analyzer 코드를 다시 객체지향적인 방식으로 재설계하는 데에 고민을 많이했습니다. Scanner 클래스 내부에서 state를 9개보다 더 많고 자세히 나누게 될 경우 프로그램이 절차지향적인 코드와 다를 것이 없을 것 같아서 **대분류와 소분류** 로 나누었습니다.
위 첨부한 State Diagram은 대분류를 의미하며 MiniCScanner 클래스에서 사용하는 분류입니다. 그리고, 각각의 심볼들을 배정할 때는 TokenString을 통해 각각의 심볼들을 배정했습니다. 이 방식을 사용하면 더욱 효율적으로 할 수 있을 것이라고 생각해서 구현했습니다.
또한 이후 컴파일러를 구현할 때 MiniCScanner 객체에서 `getToken()` 메소드만 호출하면 Token이 추출되어 Parser에서 쉽게 적용할 수 있기 때문에 모듈화가 잘되었다고 생각합니다.
이번 과제를 하면서 Lexical Analyzer가 어떻게 동작하는 지에 대해서 자세히 알 수 있었고, 설계 과정에서 State Diagram이 필요한 이유를 알게되었습니다. 처음 과제를 시작할 때 State를 나누지 않고 프로그램을 구현했더니 분기처리가 상당히 많아지게 되어 코드를 읽기 힘들었습니다. 그러나 State를 정하여 프로그램을 다시 구현하고 나니 훨씬 간결하고 읽기 쉬운 코드로 작성할 수 있었습니다.