REPORT #02

컴파일러 구성론 #02
: 한글 Compiler

과 목 명 :	컴파일러 구성론
담당교수 :	김차성 교수님
학 과 :	컴퓨터공학과
학 번 :	201200587
성 명 :	김시원
제 출 일 :	2016년 10월 06일

HANKUK UNIVERSITY OF FOREIGN STUDIES

1. 문제 정의.

제목: Automata 를 이용한 Lexical analyzer(Scanner) 만들기
제출마감: 2016년 10월 6일(목)

1. MiniJava Syntax -- Lecture-info/minijava/MiniJavaBNF.pdf(txt) 참조
2. MiniJavaLexCPP Project 참조 (input:binarytree.java, output:binarytree.tok)
3. C로 작성, lexer.c 만 완성하고 나머지는 그대로 사용

2. 해결방법 기술.

01. 문제 해결 방법과 필요 파일 및 파일 내부의 클래스와 함수를 설명, 기술하였고, 여기서 기술한 함수들은 맨 뒤에 주석처리가 된 소스코드로 기술하였다.

1. 설정에 맞게 세팅
2. 각 파일의 역할 분석
3. 본격적인 컴파일 메소드 yylex1(); 구현

우선 해결 방법 기술에 앞서 당장에 새롭게 컴파일하기위한 몇가지 설정들이 요구된다. 수업시간에 교수님께서 가르쳐주신 동영상을 참고해 세팅작업을 이루고, 후에 역할 분석을 통해 각 파일이 무슨 일을 하는지를 파악하고, 본격적인 컴파일 메소드 yylex();를 구현하였다.

1. 설정에 맞게 세팅

1. MINGW 설치

eclipse c++ 버전을 통해 지난 PL과제를 쭈욱 해왔었는데, 이때 사실 MINGW가 설정상 아무리 해도 잘 되지가 않아서 사실 cygwin이라고 하는 컴파일러를 사용했었다. 이 때문에 번거롭지만 설정을 맞추기 위해 MINGW를 다시 설치했다. 64bit 운영체제를 위한 버전이 따로 업로드 되어있어서 이를 이용하려고 했으나, msys 폴더와 그 내부에 들어있는 1.0, lib 폴더가 없는 아예 새로운 메커니즘의 MINGW라서 이 역시 설치하지 않았다.
구형 MINGW를 다운받아 환경변수 편집에서 PATH를 MINGW가 설치된 경로로 맞춰주었고, eclipse c++에서 WIndow - preference의 New C/C++ Project의 Makefile Project에서 Builder Setting의 Build command를 mingw32-make로 설정해서 OK를 하여 정상적으로 동작하는 eclipse를 확인할 수 있었다.

2. 세팅

svn의 CC2016에 있는 모든 프로젝트를 전부 받아들여온 후, 먼저 MiniJavaFLex를 실행하였지만 역시 yywrap not found가 떴다. minijava.lex를 flex로 실행하기 위한 옵션을 먼저 진행해야 하는데, 이는 먼저 External Run toos Configuration에 들어가서 수정을 해야 한다. Program 탭에서 새로운 Configuration의 이름을 지어주고, Main탭의 Location을 C:₩pub₩MinGW₩msys₩1.0₩bin₩flex.exe (MINGW가 설치된 경로의 flex.exe의 주소) 로 설정해주고, Working Directory는 FlexBisonProj 라는 교수님이 올려주신 프로젝트의 flexbison 폴더를 설정해주면 된다. Arguments는 minijava.lex를 등록해준다. 사실 이 프로젝트가 아니더라도, minijava.lex가 저장되어있는 어떤 경로를 설정해도 상관이 없다.

3. 새 프로젝트 생성

교수님의 프로젝트들을 토대로 과제를 위한 새로운 프로젝트를 생성해주어야 한다. 이름은 MyCompiler02로 설정했고 src폴더를 만들어주고, driver.c , errormsh.c, errormsg.h, lex.yy.c, lexer.c, util.c, util.h를 가져온다. 자체적으로 lex.yy.c를 통한 컴파일 결과 화면을 확인해볼 수 있게 하기위해서 tools폴더 (minijava.lex)도 복사해왔다. 이 컴파일러가 분석하기 위한 java파일인 binarytree.java도 프로젝트 경로 최상단에 복사해준다. 여기서 그냥 빌드하면 당연히 Error가 뜬다. 이유는 프로젝트에 자체 설정이 아직 남아있다.
이 프로젝트의 Propertioes 탭의 C/C++Build 탭의 Setting항목에서 MINGW C++ Linker에서 Library search path를 "C:₩pub₩MinGW₩msys₩1.0₩lib" 로 설정해주고, Libraries는 fl 로 설정해야한다. 이 "fl"의 의미는 library의 수많은 library들 중에 fl이라는 이름의 Library만을 선택적으로 뽑아 사용한다는 의미라고 수업시간에 배웠다. 이제 프로젝트를 실행해보면 lex.yy.c를 통한 컴파일이 실행이 되고, 결과는 아주 잘 나온다.
그러나 이것을 위한과제가 아니다. 지금까지 한 것은 yywrap에 의해 만들어진 화면이다 즉, minijava.lex와 lex.yy.c를 flex로 작업시켜서 이것을 통해 binarytree.java가 분석되어 나타내어진 화면이다. 이것은 아주 잘 짜여진 교본의 일부다. lex.yy.c는 java파일을 읽고, buffer에 저장하거나 읽어내는 쪽의 역할을 맡는 코드의 내용이 쓰여있고, minijava.lex는 lexical definition이 되어있다. 즉 모든 lex의 예약어나 정의되어진 기호등을 정의해놓은 파일이다. 이번 과제는 lexer.c를 이용하여 이와 같이 잘 짜여진 예약어나 정의되어진 기호, 그리고 Analyzer를 만들어야 한다. 즉 이제껏 한 작업은 MiniJavaFlex 프로젝트와 MinijavaLexCPP 프로젝트의 파일들을 모두 가져온 프로젝트를 만들어 설정을 해놓은 것으로써, lex.yy.c와 minijava.lex를 이용한 flex로 작업시키는 Flex방식과 MinijavaCPP 프로젝트의 lexer.c를 통해 Analyze하는 방식 2가지 모두 약간의 코드 변경으로 가능하게 할 수 있는 프로젝트를 만들어 놓은 것이라고 결론지을 수 있겠다. Flex방식은 완성된 방식이고, lexer.c 방식이 우리가 구현해야 할 방식인 것이다.

2. 각 파일의 역할 분석

목표가 lexer.c의 완전 구현이므로 MinijavaCPP 프로젝트에 들어있던 파일을 중심으로 기술하겠다.
먼저 tokens.h는 모든 예약어나 기호, 논리값에 대한 define이 되어있고, 그 분호는 모두 257~302까지의 값으로 구성이 되어있다. 또한 구조체와 흡사한 union이 정의되어있는데, iva, sval로 구성이 되어있다. ival은 후에 INTLIT을 출력할 때 그 해당 내용이 숫자인 경우 해당 token을 저장받아 출력시키기 위한 용도고, sval은 ID나 STRLIT을 출력할 때 출력시키기 위한 용도다. 이 구조체의 이름은 yylval이다. 이는 flex 방식에서의 minijava.lex과 그것에 대한 기호나 lex가 무엇인지까지 정의해놓은 것은 제외하고는 그 역할이 유사하다고 볼 수 있다.

errormsg.c나 errormsg.h는 파일을 열거나 열 때의 오류를 위한 메소드 그리고 파일을 읽어서 분석시에 개행을 할 때 linenum이라는 값을 증가시켜주는 기능을 가진 메소드 3가지를 정의해놓은 파일이다.
util.c, util.h는 굳이 String 라이브러리를 쓰지 않게끔 string이라는 char* 배열과 bool (진리값)을 가진 변수를 선언하였고 이 string을 출력시키기 위한 메소드와 strbuf라고 하는 char형 배열을 정의해 놓았는데 이

배열의 용도는 파일에서 읽어들인 글자가 저장되며 일종의 버퍼 역할을 한다. 이 버퍼에 글자를 삽입하기 위한 용도가 pufbuf라는 메소드이고 반대로 buf를 완전히 초기화 하기위한 clearbuf라는 메소드도 정의되어있다. bufptr이라고 하는 현 상태의 index를 저장시키는 int형 변수도 가지고 있다.
즉 이들은 모두 flex방식에서의 lex.yy.c와 그 역할이 유사하다. lex의 정의와는 별개로 파일의 입출력이나 글자의 개행, buffer등이 매우 상세하게 저장되기 위한 수많은 코드들로 구성이 되어있어 규모상 비교할수도 없을 정도지만, 그러한 역할을 수행한다는 면에서 유사하다고 볼 수 있다.

driver.c는 main 메소드가 담긴 파일로써, int yylex가 선언이 되어있는데 이는 flex방식에서 사용되는 메소드이며 우리가 사용할 메소드가 아니다. 또한 위에서 설명한 yylval이 정의 되어있다. 또 toknames[]라는 string 형 배열이 정의 되어있는데 이것은 ID, STRLIT, INTLIT,,, 등등으로 구성이 되어있고, 현재 탐색하여 조합된 token이 해당되는 lexical Type들을 모두 그 이름으로써 저장시켜놓은 배열이다.
이것은 tokens.h와 밀접한 관계를 맺고있는데, 그 이유는 다음과 같다.
위에 설명했듯이 tokens.h는 그 token이 속해질 lexical Type들을 #define으로 정의해 놓았고, 만약 token이 이 type에 해당하는 경우에는 이것이 define된 type의 번호에 맞는 고유 번호가 그대로 파라미터로써 대입될 tokname이라는 메소드가 정의되어있다는 점이다. 그러면 이 메소드에서는 해당 token이 define된 번호가 257~302사이의 번호에 해당된다면 define된 type에 해당하므로 이에 알맞은 이름을 tokenames[] 배열에서 찾아서 반환시켜준다. 그러나 해당되지 않는 외부의 범위라면 이는 파리미터로 받게된 번호가 정의되지 않은 lexical Type값으로 들어온 것이므로 Bad_Token 이라는 String을 반환시켜 주게끔 되어있다. 즉 해당하는 lexical Type이 tokname이라는 메소드를 통해 toknames배열에 정의된 이름으로 반환이 되는 것이다.

마지막으로 main 함수다. 이곳에서는 binarytree.java를 open하여 이를 yylex(); 메소드가 loop를 통해 계속 실행하며 분석한다. tok에서 반환된 값 값을 통해 switch- case 문이 정의 되어있고, ID, STRLIT, INTLIT의 경우는 각각의 lexical Type과 그 값이 출력되게끔 되어있다. 이외 나머지 값은 굳이 그 값이나 글자까지 출력할 필요는 없기 때문에 default: 로 정의하여 그 token의 lexical Type만을 출력하게 되어있다. 만약 tok에서의 결과 값이 0이면, 이는 yylex 함수 내에서 EOF을 읽는 상태 이외에는 반환되지 않는 값을 뜻하므로 모든 loop가 탈출되어 종료된다.
yylex()는 flex방식이나 lexer.c를 이용한 방식이나 모두 같은 이름의 메소드를 사용하므로 중복 오류가 된다. 이후의 기술은 편의상 flex방식은 yylex(); lexer.c방식은 yylex1();으로 명명하여 기술하겠다.

3. 본격적인 컴파일 메소드 yylex1(); 구현

token의 종류는 예약어와 Identifier, Number, String literal, Comment, 약속된 기호 즉 SPECIAL SYMBOL 등으로 구분할 수 있다. 위에 기술했듯 token이 이러한 type에 해당하는 경우에는 이것이 define된 type의 번호에 맞는 고유 번호가 그대로 파라미터로써 대입될 tokname이라는 메소드에 들어가게 되는데, 이 메소드는 파라미터로 들어간 고유 번호가 257~302사이의 번호에 해당되면 define된 type에 해당하므로 이에 알맞은 이름을 tokenames[] 배열에서 찾아서 반환시켜준다. 그러나 해당되지 않는 외부의 범위라면 이는 파리미터로 받게된 번호가 정의 되지 않은 lexical Type값으로 들어온 것이 되어 Bad_Token 이라는 String을 반환시켜 주게끔 되어있다. 그러나 주어진 yylex1(); 메소드에서는 ID를 제외한 모든 경우에 대해 정의가 되어있지 않다. 그것이 MiniJavaCPP 프로젝트를 실행시키면 ID 이외에는 BAD_TOKEN이라는 글자가 뜨는 이유다. 결과적으로 이 과제의 목표는 tokens.h에 정의된 token의 타입을 분류하기 위한 case 문을 추가하고, if문을 추가하여 올바른 식별 값을 반환시켜주는데 있다 특정 token의 타입이 값을 가지고 있다면 token의 타입뿐만 아니라 해당하는 값까지 함께 출력해 주어야 한다.

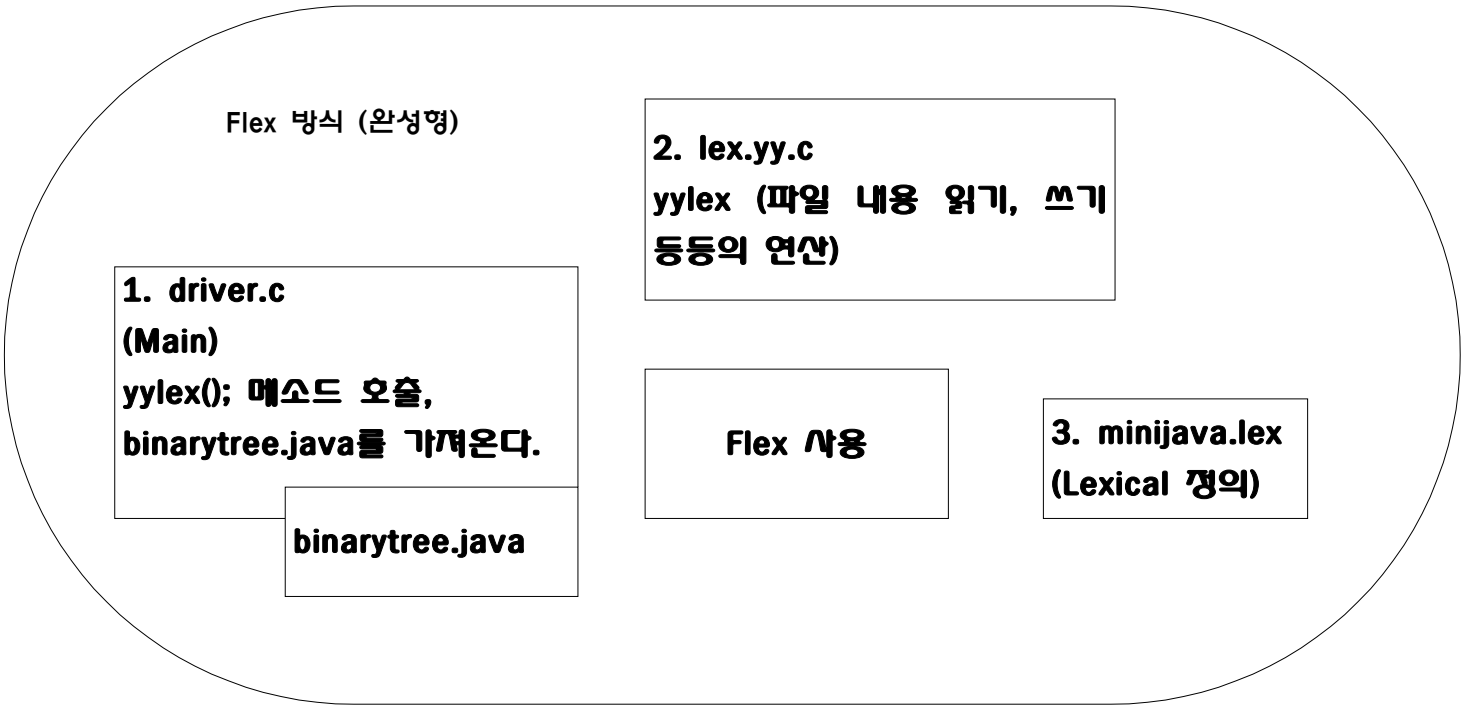
먼저 getc() 메소드를 통해 문자가 한 문자씩 읽어오도록 설정이 되어있다. 읽어온 문자는 계속적으로 버퍼에 저장되며 순간순간의 저장된 버퍼의 값이 token이라는 변수에 저장되게 되는데 이것이 우선 Number의 경우, DIGIT으로 분류 기준이 정해져 있다. 입력된 하나의 문자가 숫자라면 case 1에 해당되며 이 값은 다음 ch가 숫자가 아닌 값이 나올 때까지 계속 숫자이다. 입력된 문자가 DIGIT이라면 case 1에 해당되며 문자를 버퍼에 입력하고, 다음 문자를 읽는 작업을 반복하며 DIGIT인 경우에는 계속 입력을 받고 DIGIT이 아는 ch를 받게 되는 순간 이 조건에서는 버퍼에 입력된 값을 atoi 메소드로 int 타입으로 변환시켜 yylval의 ival 변수에 저장시키고 메인 함수인 driver.c에서 반환되는 타입에 따라 출력 형태가 정해진다.

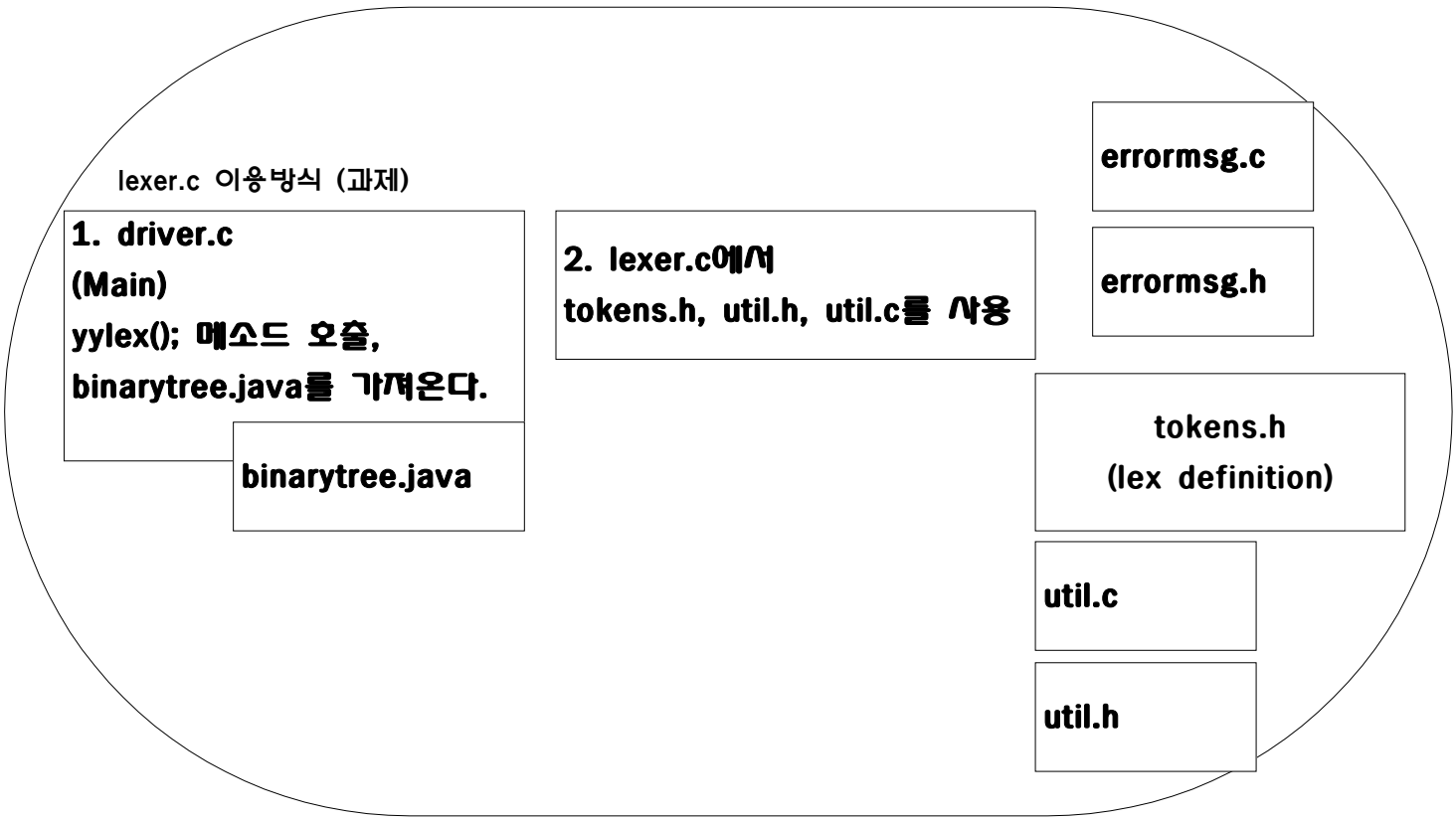
STRLIT의 경우, a-z, A-Z인 범위의 LETTER로 분류 기준이 정의되어있다. getc로 받게되는 ch가 예약어도 될수 있고 ID도 될 수 있는 경우이기 때문에 case 2에 해당하는 경우이다. 즉 이곳에 올바른 반환값을 위한 추가적인 조건문이 필요하다. 만약 버퍼에 입력되어 있는 문자열이 예약어라면 상태를 초기화하여 새로운 첫 문자를 받아 판별하기 위한 케이스 0으로 이동하게 설정하고 예약어에 해당되는 token의 이름을 반환하여 메인 함수에서 이 token의 이름이 출력되도록 한다. 예약어가 아닌 경우에는 아직 예약어로서는 미완성된 token을 지닌 경우거나 ID가 될수 있으므로 다시 판별을 하여야 한다.
즉, 읽어온 문자가 DIGIT이거나 LETTER일 경우 해당 case를 계속 반복하도록 하고 이외의 경우에는 ID로 판단하여 ID에 반환한다. 입력된 문자 중에서 공백이 나오면 지금까지 저장된 버퍼에 입력된 문자열이 token으로 지정된 예약어인지를 판별한다.

입력된 문자가 DIGIT도 LETTER도 WHITESPACE도 아니라면 이는 case3로 이동되고 이 경우는 Special symbol에 해당한다. 따라서 이 경우 하나의 문자로 Special symbol이 되는 경우는 그대로 해당되는 token의 타입으로 반환하여 준다. 또한 " 와 같이 문장의 시작을 알리는 Symbol은 반환되지 않고 끝을 알기 위해 state 6로 설정되어 case6로 보내진다. 이곳에서는 "로 시작하는 심볼을 확인하여 다음 "를 만날 때까지 읽어 들이는 문자를 버퍼에 입력하여 다음 "를 만났을 때 STRING을 반환하여 버퍼의 값을 출력한다. 또한 문자 중에 ₩가 있다면 다음 문자가 "라고 하더라도 그대로 문자열에 포함되는 "로 간주하므로 ₩가 입력되었을 때 다음 문자를 읽어 무조건 버퍼에 저장하여 다음 "를 만날 때까지 하나의 문자열이 될 수 있도록 한다. case3 의 남은 경우는 바로 =나 !와 같이 다음 ch가 합쳐 졌을 때 완전이 다른 Symbol로 조합되어질 수 있는 경우인데 이는 마치 지난 과제의 J, JM JMM 과 같이, 우선 저장되어져 있는 상태로 다른 case로 이동되어 그 경과를 지켜보야야 한다. 그래서 case 4를 만들어 다음 문자까지 비교하여 두 개의 문자가 또다른 symbol이 될 수 있는지를 판단한다.

이러한 case4의 경우는 내부에서 다시한번 더 조건연산이 수행된다. 추가적으로 얻어낸 ch를 조합하여 이 token이 2개의 ch를 합쳐서 만들어진 Symbol인 경우의 조건은 해당 Symbol의 값이 반환되고 case0으로 이동되어 계속 진행된다. 이외의 경우는 추가로 얻은 ch가 합쳐져도 다른 Symbol이 되지 못하는 경우이므로 먼저 case3에서 얻은 상태 그대로 반환되며 sate는 0이 된다. 이외의 경우는 WHITESPACE이거나 token에 정의되지 않은 문자이므로 그 문자를 그대로 출력하도록 한다. 추가적으로 얻어낸 ch를 얻었을 때 2개의 ch가 합쳐져서 Symbol이 되었어도 다시 한번 더 다른 State로 가게되는 두 가지의 Symbol이 있는데 그것은 바로 주석에 관련된 Symbol이다. 긴 주석의 시작 Symbol인 /*가 입력된 경우에는 새로운 case 5를 만들고 state=5로 두어 진입시킨다. 이 case5에서는 주석의 끝을 의미하는 */가 나올 때까지 계속해서 문자를 읽어 들이고 그렇지 않은 경우는 버려도 되는 경우이므로 읽어가며 계속 버퍼를 비워주면서 주석이 끝남과 동시에 주석이 모두 날아가 주석은 출력되지 않도록 한다. 만약 //이 나오면 이는 한줄 주석이므로 새로운 case 7를 만들고 이곳에 진입시킨다. 이 case에서는 개행이 나오는 경우까지는 문자를 읽어 들이고 그렇지 않은 경우는 버려도 되는 경우이므로 읽어가며 계속 버퍼를 비워주면서 주석이 끝남과 동시에 주석이 모두 날아가 주석은 출력되지 않도록 한다.

3. 프로그램 구성도.


Flex 방식 (완성형)
2. lex.yy.c
yylex (파일 내용 읽기, 쓰기 등등의 연산)
1. driver.c
(Main)
yylex(); 메소드 호출,
binarytree.java를 가져온다.
Flex 사용
3. minijava.lex
(Lexical 정의)
binarytree.java


lexer.c 이용방식 (과제)
errormsg.c
1. driver.c
(Main)
yylex(); 메소드 호출,
binarytree.java를 가져온다.
2. lexer.c에서
tokens.h, util.h, util.c를 사용
errormsg.h
tokens.h
(lex definition)
binarytree.java
util.c
util.h

4. 프로그램 실행 및 출력 예.

3단으로 나뉘어 있는데, 1페이지 당 3단형태로 분류되어있습니다. 1단이 다른쪽으로 이어져 있는 것이 아닙니다. 1단에서 2단은 왼쪽에서 오른쪽으로 넘어갑니다.

CLASS
ID (BinaryTree)
LBRACE
PUBLIC
STATIC
VOID
MAIN
LPAREN
STRING
LBRACK
RBRACK
ID (a)
RPAREN
LBRACE
ID (System)
DOT
ID (out)
DOT
ID (println)
LPAREN
STRLIT (MiniJava BinaryTree Main)
RPAREN
SEMICOLON
ID (System)
DOT
ID (out)
DOT
ID (println)
LPAREN
NEW
ID (BT)
LPAREN
RPAREN
DOT
ID (Start)
LPAREN
RPAREN
RPAREN
SEMICOLON
RBRACE
RBRACE
CLASS
ID (BT)
LBRACE
PUBLIC
INTEGER
ID (Start)
LPAREN
RPAREN
LBRACE
ID (Tree)
ID (root)
SEMICOLON
BOOLEAN
ID (ntb)
SEMICOLON
INTEGER
ID (nti)
SEMICOLON
ID (root)
ASSIGN
NEW
ID (Tree)
LPAREN
RPAREN
SEMICOLON
ID (ntb)
ASSIGN
ID (root)
DOT
ID (Init)
LPAREN
INTLIT (16)
RPAREN
SEMICOLON
ID (ntb)
ASSIGN
ID (root)
DOT
ID (Print)
LPAREN
RPAREN
SEMICOLON
ID (System)
DOT
ID (out)
DOT
ID (println)
LPAREN
INTLIT (100000000)
RPAREN
SEMICOLON
ID (ntb)
ASSIGN
ID (root)
DOT
ID (Insert)
LPAREN
INTLIT (8)
RPAREN
SEMICOLON
ID (ntb)
ASSIGN
ID (root)
DOT
ID (Print)
LPAREN
RPAREN
SEMICOLON
ID (ntb)
ASSIGN
ID (root)
DOT
ID (Insert)
LPAREN
INTLIT (24)
RPAREN
SEMICOLON
ID (ntb)
ASSIGN
ID (root)
DOT
ID (Insert)
LPAREN
INTLIT (4)
RPAREN
SEMICOLON
ID (ntb)
ASSIGN
ID (root)

DOT
ID (Insert)
LPAREN
INTLIT (12)
RPAREN
SEMICOLON
ID (ntb)
ASSIGN
ID (root)
DOT
ID (Insert)
LPAREN
INTLIT (20)
RPAREN
SEMICOLON
ID (ntb)
ASSIGN
ID (root)
DOT
ID (Insert)
LPAREN
INTLIT (28)
RPAREN
SEMICOLON
ID (ntb)
ASSIGN
ID (root)
DOT
ID (Insert)
LPAREN
INTLIT (14)
RPAREN
SEMICOLON
ID (ntb)
ASSIGN
ID (root)
DOT
ID (Print)
LPAREN
RPAREN
SEMICOLON
ID (System)
DOT
ID (out)
DOT
ID (println)
LPAREN
ID (root)
DOT
ID (Search)
LPAREN
INTLIT (24)
RPAREN
RPAREN
SEMICOLON
ID (System)
DOT
ID (out)
DOT
ID (println)
LPAREN
ID (root)
DOT
ID (Search)
LPAREN
INTLIT (12)
RPAREN
RPAREN
SEMICOLON
ID (System)
DOT
ID (out)
DOT
ID (println)
LPAREN
ID (root)
DOT
ID (Search)
LPAREN
INTLIT (16)
RPAREN
RPAREN
SEMICOLON
ID (System)
DOT
ID (out)
DOT
ID (println)
LPAREN
ID (root)
DOT
ID (Search)
LPAREN
INTLIT (50)
RPAREN
RPAREN
SEMICOLON
ID (System)
DOT
ID (out)
DOT
ID (println)
LPAREN
ID (root)
DOT
ID (Search)
LPAREN
INTLIT (12)
RPAREN
RPAREN
SEMICOLON
ID (ntb)
ASSIGN
ID (root)
DOT
ID (Delete)
LPAREN
INTLIT (12)
RPAREN
SEMICOLON
ID (ntb)
ASSIGN
ID (root)
DOT
ID (Print)
LPAREN
RPAREN
SEMICOLON
ID (System)
DOT
ID (out)
DOT
ID (println)
LPAREN
ID (root)
DOT
ID (Search)
LPAREN
INTLIT (12)
RPAREN
RPAREN
SEMICOLON
RETURN
INTLIT (0)
SEMICOLON
RBRACE
RBRACE
CLASS
ID (Tree)
LBRACE

ID (Tree)
ID (left)
SEMICOLON
ID (Tree)
ID (right)
SEMICOLON
INTEGER
ID (key)
SEMICOLON
BOOLEAN
ID (has_left)
SEMICOLON
BOOLEAN
ID (has_right)
SEMICOLON
ID (Tree)
ID (my_null)
SEMICOLON
PUBLIC
BOOLEAN
ID (Init)
LPAREN
INTEGER
ID (v_key)
RPAREN
LBRACE
ID (key)
ASSIGN
ID (v_key)
SEMICOLON
ID (has_left)
ASSIGN
FALSE
SEMICOLON
ID (has_right)
ASSIGN
FALSE
SEMICOLON
RETURN
TRUE
SEMICOLON
RBRACE
PUBLIC
BOOLEAN
ID (SetRight)
LPAREN
ID (Tree)
ID (rn)
RPAREN
LBRACE
ID (right)
ASSIGN
ID (rn)
SEMICOLON
RETURN
TRUE
SEMICOLON
RBRACE
PUBLIC
BOOLEAN
ID (SetLeft)
LPAREN
ID (Tree)
ID (ln)
RPAREN
LBRACE
ID (left)
ASSIGN
ID (ln)
SEMICOLON
RETURN
TRUE
SEMICOLON
RBRACE
PUBLIC
ID (Tree)
ID (GetRight)
LPAREN
RPAREN
LBRACE
RETURN
ID (right)
SEMICOLON
RBRACE
PUBLIC
ID (Tree)
ID (GetLeft)
LPAREN
RPAREN
LBRACE
RETURN
ID (left)
SEMICOLON
RBRACE
PUBLIC
INTEGER
ID (GetKey)
LPAREN
RPAREN
LBRACE
RETURN
ID (key)
SEMICOLON
RBRACE
PUBLIC
BOOLEAN
ID (SetKey)
LPAREN
INTEGER
ID (v_key)
RPAREN
LBRACE
ID (key)
ASSIGN
ID (v_key)
SEMICOLON
RETURN
TRUE
SEMICOLON
RBRACE
PUBLIC
BOOLEAN
ID (GetHas_Right)
LPAREN
RPAREN
LBRACE
RETURN
ID (has_right)
SEMICOLON
RBRACE
PUBLIC
BOOLEAN
ID (GetHas_Left)
LPAREN
RPAREN
LBRACE
RETURN
ID (has_left)
SEMICOLON
RBRACE
PUBLIC
BOOLEAN
ID (SetHas_Left)
LPAREN
BOOLEAN
ID (val)
RPAREN
LBRACE
ID (has_left)
ASSIGN

ID (val)
SEMICOLON
RETURN
TRUE
SEMICOLON
RBRACE
PUBLIC
BOOLEAN
ID (SetHas_Right)
LPAREN
BOOLEAN
ID (val)
RPAREN
LBRACE
ID (has_right)
ASSIGN
ID (val)
SEMICOLON
RETURN
TRUE
SEMICOLON
RBRACE
PUBLIC
BOOLEAN
ID (Compare)
LPAREN
INTEGER
ID (num1)
COMMA
INTEGER
ID (num2)
RPAREN
LBRACE
BOOLEAN
ID (ntb)
SEMICOLON
INTEGER
ID (nti)
SEMICOLON
ID (ntb)
ASSIGN
FALSE
SEMICOLON
ID (nti)
ASSIGN
ID (num2)
PLUS
INTLIT (1)
SEMICOLON
IF
LPAREN
ID (num1)
LT
ID (num2)
RPAREN
ID (ntb)
ASSIGN
FALSE
SEMICOLON
ELSE
IF
LPAREN
NOT
LPAREN
ID (num1)
LT
ID (nti)
RPAREN
RPAREN
ID (ntb)
ASSIGN
FALSE
SEMICOLON
ELSE
ID (ntb)
ASSIGN
TRUE
SEMICOLON
RETURN
ID (ntb)
SEMICOLON
RBRACE
PUBLIC
BOOLEAN
ID (Insert)
LPAREN
INTEGER
ID (v_key)
RPAREN
LBRACE
ID (Tree)
NEW
ID (_node)
SEMICOLON
BOOLEAN
ID (ntb)
SEMICOLON
BOOLEAN
ID (cont)
SEMICOLON
INTEGER
ID (key_aux)
SEMICOLON
ID (Tree)
ID (current_node)
SEMICOLON
NEW
ID (_node)
ASSIGN
NEW
ID (Tree)
LPAREN
RPAREN
SEMICOLON
ID (ntb)
ASSIGN
NEW
ID (_node)
DOT
ID (Init)
LPAREN
ID (v_key)
RPAREN
SEMICOLON
ID (current_node)
ASSIGN
THIS
SEMICOLON
ID (cont)
ASSIGN
TRUE
SEMICOLON
WHILE
LPAREN
ID (cont)
RPAREN
LBRACE
ID (key_aux)
ASSIGN
ID (current_node)
DOT
ID (GetKey)
LPAREN
RPAREN
SEMICOLON
IF
LPAREN
ID (v_key)
LT
ID (key_aux)

RPAREN
LBRACE
IF
LPAREN
ID (current_node)
DOT
ID (GetHas_Left)
LPAREN
RPAREN
RPAREN
ID (current_node)
ASSIGN
ID (current_node)
DOT
ID (GetLeft)
LPAREN
RPAREN
SEMICOLON
ELSE
LBRACE
ID (cont)
ASSIGN
FALSE
SEMICOLON
ID (ntb)
ASSIGN
ID (current_node)
DOT
ID (SetHas_Left)
LPAREN
TRUE
RPAREN
SEMICOLON
ID (ntb)
ASSIGN
ID (current_node)
DOT
ID (SetLeft)
LPAREN
NEW
ID (_node)
RPAREN
SEMICOLON
RBRACE
RBRACE
ELSE
LBRACE
IF
LPAREN
ID (current_node)
DOT
ID (GetHas_Right)
LPAREN
RPAREN
RPAREN
ID (current_node)
ASSIGN
ID (current_node)
DOT
ID (GetRight)
LPAREN
RPAREN
SEMICOLON
ELSE
LBRACE
ID (cont)
ASSIGN
FALSE
SEMICOLON
ID (ntb)
ASSIGN
ID (current_node)
DOT
ID (SetHas_Right)
LPAREN
TRUE
RPAREN
SEMICOLON
ID (ntb)
ASSIGN
ID (current_node)
DOT
ID (SetRight)
LPAREN
NEW
ID (_node)
RPAREN
SEMICOLON
RBRACE
RBRACE
RBRACE
RETURN
TRUE
SEMICOLON
RBRACE
PUBLIC
BOOLEAN
ID (Delete)
LPAREN
INTEGER
ID (v_key)
RPAREN
LBRACE
ID (Tree)
ID (current_node)
SEMICOLON
ID (Tree)
ID (parent_node)
SEMICOLON
BOOLEAN
ID (cont)
SEMICOLON
BOOLEAN
ID (found)
SEMICOLON
BOOLEAN
ID (is_root)
SEMICOLON
INTEGER
ID (key_aux)
SEMICOLON
BOOLEAN
ID (ntb)
SEMICOLON
ID (current_node)
ASSIGN
THIS
SEMICOLON
ID (parent_node)
ASSIGN
THIS
SEMICOLON
ID (cont)
ASSIGN
TRUE
SEMICOLON
ID (found)
ASSIGN
FALSE
SEMICOLON
ID (is_root)
ASSIGN
TRUE
SEMICOLON
WHILE
LPAREN
ID (cont)
RPAREN
LBRACE
ID (key_aux)

ASSIGN
ID (current_node)
DOT
ID (GetKey)
LPAREN
RPAREN
SEMICOLON
IF
LPAREN
ID (v_key)
LT
ID (key_aux)
RPAREN
IF
LPAREN
ID (current_node)
DOT
ID (GetHas_Left)
LPAREN
RPAREN
RPAREN
LBRACE
ID (parent_node)
ASSIGN
ID (current_node)
SEMICOLON
ID (current_node)
ASSIGN
ID (current_node)
DOT
ID (GetLeft)
LPAREN
RPAREN
SEMICOLON
RBRACE
ELSE
ID (cont)
ASSIGN
FALSE
SEMICOLON
ELSE
IF
LPAREN
ID (key_aux)
LT
ID (v_key)
RPAREN
IF
LPAREN
ID (current_node)
DOT
ID (GetHas_Right)
LPAREN
RPAREN
RPAREN
LBRACE
ID (parent_node)
ASSIGN
ID (current_node)
SEMICOLON
ID (current_node)
ASSIGN
ID (current_node)
DOT
ID (GetRight)
LPAREN
RPAREN
SEMICOLON
RBRACE
ELSE
ID (cont)
ASSIGN
FALSE
SEMICOLON
ELSE
LBRACE
IF
LPAREN
ID (is_root)
RPAREN
IF
LPAREN
LPAREN
NOT
ID (current_node)
DOT
ID (GetHas_Right)
LPAREN
RPAREN
RPAREN
AND
LPAREN
NOT
ID (current_node)
DOT
ID (GetHas_Left)
LPAREN
RPAREN
RPAREN
RPAREN
ID (ntb)
ASSIGN
TRUE
SEMICOLON
ELSE
ID (ntb)
ASSIGN
THIS
DOT
ID (Remove)
LPAREN
ID (parent_node)
COMMA
ID (current_node)
RPAREN
SEMICOLON
ELSE
ID (ntb)
ASSIGN
THIS
DOT
ID (Remove)
LPAREN
ID (parent_node)
COMMA
ID (current_node)
RPAREN
SEMICOLON
ID (found)
ASSIGN
TRUE
SEMICOLON
ID (cont)
ASSIGN
FALSE
SEMICOLON
RBRACE
ID (is_root)
ASSIGN
FALSE
SEMICOLON
RBRACE
RETURN
ID (found)
SEMICOLON
RBRACE
PUBLIC
BOOLEAN
ID (Remove)
LPAREN

ID (Tree)
ID (p_node)
COMMA
ID (Tree)
ID (c_node)
RPAREN
LBRACE
BOOLEAN
ID (ntb)
SEMICOLON
INTEGER
ID (auxkey1)
SEMICOLON
INTEGER
ID (auxkey2)
SEMICOLON
IF
LPAREN
ID (c_node)
DOT
ID (GetHas_Left)
LPAREN
RPAREN
RPAREN
ID (ntb)
ASSIGN
THIS
DOT
ID (RemoveLeft)
LPAREN
ID (p_node)
COMMA
ID (c_node)
RPAREN
SEMICOLON
ELSE
IF
LPAREN
ID (c_node)
DOT
ID (GetHas_Right)
LPAREN
RPAREN
RPAREN
ID (ntb)
ASSIGN
THIS
DOT
ID (RemoveRight)
LPAREN
ID (p_node)
COMMA
ID (c_node)
RPAREN
SEMICOLON
ELSE
LBRACE
ID (auxkey1)
ASSIGN
ID (c_node)
DOT
ID (GetKey)
LPAREN
RPAREN
SEMICOLON
ID (auxkey2)
ASSIGN
LPAREN
ID (p_node)
DOT
ID (GetLeft)
LPAREN
RPAREN
RPAREN
DOT
ID (GetKey)
LPAREN
RPAREN
SEMICOLON
IF
LPAREN
THIS
DOT
ID (Compare)
LPAREN
ID (auxkey1)
COMMA
ID (auxkey2)
RPAREN
RPAREN
LBRACE
ID (ntb)
ASSIGN
ID (p_node)
DOT
ID (SetLeft)
LPAREN
ID (my_null)
RPAREN
SEMICOLON
ID (ntb)
ASSIGN
ID (p_node)
DOT
ID (SetHas_Left)
LPAREN
FALSE
RPAREN
SEMICOLON
RBRACE
ELSE
LBRACE
ID (ntb)
ASSIGN
ID (p_node)
DOT
ID (SetRight)
LPAREN
ID (my_null)
RPAREN
SEMICOLON
ID (ntb)
ASSIGN
ID (p_node)
DOT
ID (SetHas_Right)
LPAREN
FALSE
RPAREN
SEMICOLON
RBRACE
RBRACE
RETURN
TRUE
SEMICOLON
RBRACE
PUBLIC
BOOLEAN
ID (RemoveRight)
LPAREN
ID (Tree)
ID (p_node)
COMMA
ID (Tree)
ID (c_node)
RPAREN
LBRACE
BOOLEAN
ID (ntb)
SEMICOLON

WHILE
LPAREN
ID (c_node)
DOT
ID (GetHas_Right)
LPAREN
RPAREN
RPAREN
LBRACE
ID (ntb)
ASSIGN
ID (c_node)
DOT
ID (SetKey)
LPAREN
LPAREN
ID (c_node)
DOT
ID (GetRight)
LPAREN
RPAREN
RPAREN
DOT
ID (GetKey)
LPAREN
RPAREN
RPAREN
SEMICOLON
ID (p_node)
ASSIGN
ID (c_node)
SEMICOLON
ID (c_node)
ASSIGN
ID (c_node)
DOT
ID (GetRight)
LPAREN
RPAREN
SEMICOLON
RBRACE
ID (ntb)
ASSIGN
ID (p_node)
DOT
ID (SetRight)
LPAREN
ID (my_null)
RPAREN
SEMICOLON
ID (ntb)
ASSIGN
ID (p_node)
DOT
ID (SetHas_Right)
LPAREN
FALSE
RPAREN
SEMICOLON
RETURN
TRUE
SEMICOLON
RBRACE
PUBLIC
BOOLEAN
ID (RemoveLeft)
LPAREN
ID (Tree)
ID (p_node)
COMMA
ID (Tree)
ID (c_node)
RPAREN
LBRACE
BOOLEAN
ID (ntb)
SEMICOLON
WHILE
LPAREN
ID (c_node)
DOT
ID (GetHas_Left)
LPAREN
RPAREN
RPAREN
LBRACE
ID (ntb)
ASSIGN
ID (c_node)
DOT
ID (SetKey)
LPAREN
LPAREN
ID (c_node)
DOT
ID (GetLeft)
LPAREN
RPAREN
RPAREN
DOT
ID (GetKey)
LPAREN
RPAREN
RPAREN
SEMICOLON
ID (p_node)
ASSIGN
ID (c_node)
SEMICOLON
ID (c_node)
ASSIGN
ID (c_node)
DOT
ID (GetLeft)
LPAREN
RPAREN
SEMICOLON
RBRACE
ID (ntb)
ASSIGN
ID (p_node)
DOT
ID (SetLeft)
LPAREN
ID (my_null)
RPAREN
SEMICOLON
ID (ntb)
ASSIGN
ID (p_node)
DOT
ID (SetHas_Left)
LPAREN
FALSE
RPAREN
SEMICOLON
RETURN
TRUE
SEMICOLON
RBRACE
PUBLIC
INTEGER
ID (Search)
LPAREN
INTEGER
ID (v_key)
RPAREN
LBRACE
BOOLEAN
ID (cont)

SEMICOLON
INTEGER
IF
ID (ound)
SEMICOLON
ID (Tree)
ID (current_node)
SEMICOLON
INTEGER
ID (key_aux)
SEMICOLON
ID (current_node)
ASSIGN
THIS
SEMICOLON
ID (cont)
ASSIGN
TRUE
SEMICOLON
IF
ID (ound)
ASSIGN
INTLIT (0)
SEMICOLON
WHILE
LPAREN
ID (cont)
RPAREN
LBRACE
ID (key_aux)
ASSIGN
ID (current_node)
DOT
ID (GetKey)
LPAREN
RPAREN
SEMICOLON
IF
LPAREN
ID (v_key)
LT
ID (key_aux)
RPAREN
IF
LPAREN
ID (current_node)
DOT
ID (GetHas_Left)
LPAREN
RPAREN
RPAREN
ID (current_node)
ASSIGN
ID (current_node)
DOT
ID (GetLeft)
LPAREN
RPAREN
SEMICOLON
ELSE
ID (cont)
ASSIGN
FALSE
SEMICOLON
ELSE
IF
LPAREN
ID (key_aux)
LT
ID (v_key)
RPAREN
IF
LPAREN
ID (current_node)
DOT
ID (GetHas_Right)
LPAREN
RPAREN
RPAREN
ID (current_node)
ASSIGN
ID (current_node)
DOT
ID (GetRight)
LPAREN
RPAREN
SEMICOLON
ELSE
ID (cont)
ASSIGN
FALSE
SEMICOLON
ELSE
LBRACE
IF
ID (ound)
ASSIGN
INTLIT (1)
SEMICOLON
ID (cont)
ASSIGN
FALSE
SEMICOLON
RBRACE
RBRACE
RETURN
IF
ID (ound)
SEMICOLON
RBRACE
PUBLIC
BOOLEAN
ID (Print)
LPAREN
RPAREN
LBRACE
ID (Tree)
ID (current_node)
SEMICOLON
BOOLEAN
ID (ntb)
SEMICOLON
ID (current_node)
ASSIGN
THIS
SEMICOLON
ID (ntb)
ASSIGN
THIS
DOT
ID (RecPrint)
LPAREN
ID (current_node)
RPAREN
SEMICOLON
RETURN
TRUE
SEMICOLON
RBRACE
PUBLIC
BOOLEAN
ID (RecPrint)
LPAREN
ID (Tree)
ID (node)
RPAREN
LBRACE
BOOLEAN
ID (ntb)
SEMICOLON

IF
LPAREN
ID (node)
DOT
ID (GetHas_Left)
LPAREN
RPAREN
RPAREN
LBRACE
ID (ntb)
ASSIGN
THIS
DOT
ID (RecPrint)
LPAREN
ID (node)
DOT
ID (GetLeft)
LPAREN
RPAREN
RPAREN
SEMICOLON
RBRACE
ELSE
ID (ntb)
ASSIGN
TRUE
SEMICOLON
ID (System)
DOT
ID (out)
DOT
ID (println)
LPAREN
ID (node)
DOT
ID (GetKey)
LPAREN
RPAREN
RPAREN
SEMICOLON
IF
LPAREN
ID (node)
DOT
ID (GetHas_Right)
LPAREN
RPAREN
RPAREN
LBRACE
ID (ntb)
ASSIGN
THIS
DOT
ID (RecPrint)
LPAREN
ID (node)
DOT
ID (GetRight)
LPAREN
RPAREN
RPAREN
SEMICOLON
RBRACE
ELSE
ID (ntb)
ASSIGN
TRUE
SEMICOLON
RETURN
TRUE
SEMICOLON
RBRACE
RBRACE

6. 느낀점 및 소감.

'문자를 문자자체로 해석하면 얼마나 좋을까? 아버지를 아버지라고 부르지 못하는 것 같네' 컴파일러가 필요 없는 세상을 부르게 될지도 모르는 이러한 생각을 잠시 해봤다. 컴퓨터는 어떤 언어의 문자를 문자 자체로 읽지 못하고 2진법과 0, 1로만 구성되어있는 세계로 끌어들여 분석한다. 그러다보니 논리와 수학적 산술을 통한 판독으로 토큰별로 분류하고 이해한다. 그래서인지 컴퓨터 속에서의 언어체계는 참으로 과학적이고 감탄스럽다. 새삼 프로그래밍 언어를 만든 사람들이 대단해 보인다. 컴파일러 구성론의 학문적인 의의는 정말 프로그래밍을 위한 학문이기보다는 컴퓨터 자체에서의 문자를 처리하는 방식을 배우는 것과 가까운 것 같다. 컴퓨터가 나온 이례 프로그래밍이라는 개념 이전부터 다른 분야에서도 수없이 고민하고 연구해온 학문일 것이다. 여태껏 배웠던 프로그래밍 학문보다 훨씬 Low Level 적인 요소를 가지고 있다고 생각한다. 이번 과제는 이러한 느낌을 그 어느 때보다도 제대로 느끼게 되는 과제였다. 이러한 면에서는 사람은 정말 컴퓨터에 비하면 똑똑한 건가 싶다. 과제에서의 가장 어려웠던 점은 토큰의 분류 기준이 명확하지 않았다는 점이다. 그래서 이에 대한 분류 기준을 정하고 이에 해당하는 큰 조건을 만들어, 그 조건 내부에서 점점 세부적인 조건으로 파고드는 형식으로 코드를 짜나갔던 것 같다. 점진적인 분류 때문인지 시간이 너무 많이 걸렸다. 특히 주석에 대한 케이스를 따로 분류하여 처리하는 방식은 정말로 떠올리는데 큰 고생이 있었다. 주석도 /* 방식과 // 방식이 분류되어있어서 그 케이스도 별개로 분류시켰다. 또, 단일 Symbol이 아닌 2개의 character가 조합된 Symbol을 만드는 조건의 분기를 어떻게 처리하면 좋을지 감이 안잡혔었는데 다행히도 이점은 지난 첫 번째 과제에서의 도움이 컸던 것 같다.