Design and Development of Compiler  
for C- Language

**Phase I**

**Design and Implementation**

**of Lexical Analyzer**

**Project Result**

과목명: CSE4120 기초 컴파일러 구성

담당 교수: 서강대학교 컴퓨터공학과 정성원

개발자: 20141589 최광희

제출일: 2017. 3. 27.

1. **개발 목표**

전체 C- Compiler 설계 프로젝트 중 Phase I의 개발 목표는 Lexical Analyzing of C- Language로, C- Language로 쓰인 소스 코드를 Token 단위로 쪼개 분석할 수 있도록 한다.

1. **개발 범위 및 내용**

**1. 개발 범위**

Lexical Analysis는 주어진 C- 언어 소스 코드에 대해 각 Token별로 쪼개기 위해, 각 Token을 나타내는 Regular Expression을 통해 생성한 Automation을 모조리 묶어 하나의 Deterministic Finite State Automation을 만드는 것을 말한다. 이 과정에서 Regular Expression으로부터 Automation을 생성하는 과정, Nondeterministic Finite State Automation을 Deterministic Finite State Automation으로 변환하는 것 등의 과정은 flex라는 Software가 대신 생성하고, 이번 Phase에서는 오직 flex가 이해할 수 있는 Lexical Specification 소스 코드를 작성한 뒤, flex가 해당 코드를 변환시켜준 lex.yy.c를 통해 주어진 소스 코드를 Token 단위로 쪼개 출력할 수 있도록 한다.

**2. 개발 내용**

C- 언어는 C 언어의 Subset으로써, 단순한 문법 구조를 갖고 있다. Keyword는 총 6개, else / if / int / return / void / while 이 존재한다. 특수 Symbol들의 경우 + - \* / < <= > >= == != = , . ( ) [ ] { } /\* \*/ 가 있다. ID와 NUM이라는 Token이 존재하는데, ID는 오직 Alphabet, NUM은 오직 0~9 사이의 숫자로만 되어 있다. White space는 seperation 이외에는 의미를 갖지 않고, 주석의 경우 /\* \*/ 형태로, 흔히 C에서 사용하는 형태로 되어 있다.

이번 Phase에서는 먼저, 해당 언어의 분석을 통하여 BNF Grammar로 변환시키고, 해당 Grammar을 바탕으로 flex가 이해할 수 있는 Lexical Specification 소스 코드를 작성한다.

위를 완성한 뒤에는, flex가 변환을 한 뒤에 나오는 결과인 lex.yy.c 파일에 생성된 yylex() 함수를 이용하여, C- 언어로 이뤄진 소스 코드를 Lexically Analyze한다. 이 과정에서 교재에 주어진 main.c , global.h , util.h , util.c 를 적절히 작성하여 Lexical Analyzer을 설계한다.

그 과정에서 필요한 C- 언어로 이뤄진 소스 코드 역시 작성한다.

1. **추진 일정 및 개발 방법**

**1. 추진 일정**

전반적인 일정은 다음과 같다.

|  |
| --- |
| 3/18 ~ 3/20: dotfiles, git @ Github 등의 개발환경 세팅  3/20 ~ 3/21: Makefile 작성 및 Lexical Specification 설계, Lexical Analyzer 설계  3/22 ~ 3/27: Comment 예외 등 여러 Case 처리 및 Test case 작성 |

예상보다 훨씬 빠르게 개발할 수 있었는데, 대부분의 코드가 이미 교과서에 설계되어 있었기 때문이다. 또한, Travis CI는 deploy하지 않았는데, 이후 유의미한 Test가 가능해질 때 추가하고자 한다.

각 Commit 단위로 쪼갠 구체적인 일정은 다음과 같다.

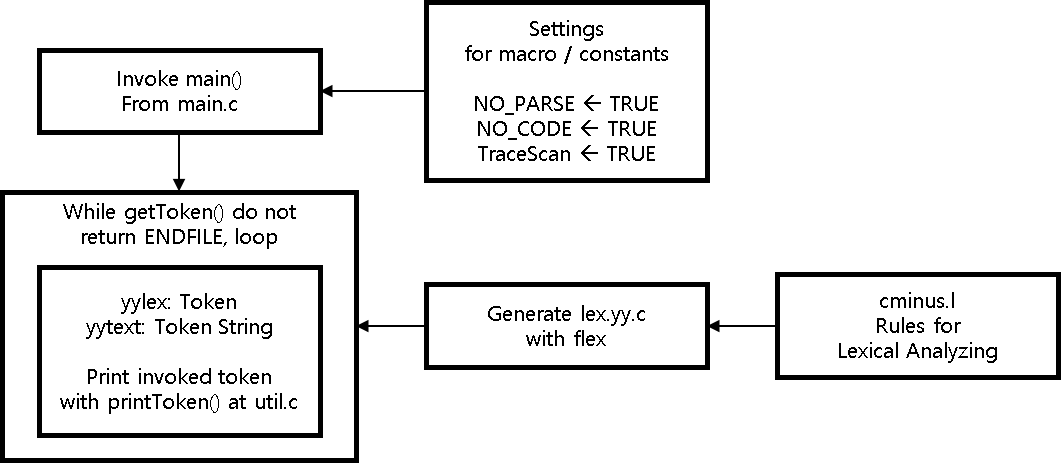
|  |
| --- |
| commit 3265802a4e460e754b9d225594074f2db065e3cd  Author: Kwanghee Choi <juice500ml@gmail.com>  Date: Tue Mar 21 03:37:22 2017 -0700  Commit inital code from subject textbook    This code is from "Compiler Construction: Principles and Practice" by Kenneth C. Louden.  commit 55c3401f926fbcc3c2af8d1471210cc5bccdcb83  Author: Kwanghee Choi <juice500ml@gmail.com>  Date: Tue Mar 21 04:40:26 2017 -0700  Commit additional code from subject textbook    This code is from "Compiler Construction: Principles and Practice" by Kenneth C. Louden.  commit d4a37a9f2589b166fa630ce633ab446cfd2e0e0b  Author: Kwanghee Choi <juice500ml@gmail.com>  Date: Tue Mar 21 04:42:21 2017 -0700  Build by default  commit 77d77662e25a84f7d1d6ae1c73cd64fcec2c0508  Author: Kwanghee Choi <juice500ml@gmail.com>  Date: Tue Mar 21 05:03:52 2017 -0700  Reindent code by GNU C Style  commit d2ee11c8fd2e5dcd6d220fbdb4a1dc9855bbee93  Author: Kwanghee Choi <juice500ml@gmail.com>  Date: Tue Mar 21 05:19:08 2017 -0700  Change TINY Lex to C- Lex    File is modified from TINY Lex file, which was given, to C- Lex  file. Reference can be found at Appendix A.1 Lexical Conventions  of C-. Reserved keywords and special symbols are added.  Modification credit is stated at the start of the modified file.  commit 3018864b3610e64f9951a29554b5f8bccd4f8947  Author: Kwanghee Choi <juice500ml@gmail.com>  Date: Tue Mar 21 05:33:04 2017 -0700  [minor] change location of comment  commit eaa10997881f72f9d6c07617b589bd873760f58a  Author: Kwanghee Choi <juice500ml@gmail.com>  Date: Tue Mar 21 05:34:52 2017 -0700  Update util.c to serve C- instead of TINY    Function printToken was developed primarily for TINY. Modified with  regard to enum TokenType at globals.h .  commit 2646de0148c32972c478096101fdf670d0063111  Author: Kwanghee Choi <juice500ml@gmail.com>  Date: Tue Mar 21 05:42:11 2017 -0700  [minor] Changes comment from TINY to C-  commit b7ae401a275dd4ecf74ccf555a7817bbdf741a0a  Author: Kwanghee Choi <juice500ml@gmail.com>  Date: Tue Mar 21 05:55:23 2017 -0700  [minor] Rename tiny.l to cminus.l  commit 0f58ade700f643579c4d7d8cce40a2fcecb0093d  Author: Kwanghee Choi <juice500ml@gmail.com>  Date: Tue Mar 21 05:59:15 2017 -0700  [minor] Normalize end-of-line  commit f2198b537f6120ae659af9d5a247f6b1142e275f  Author: Kwanghee Choi <juice500ml@gmail.com>  Date: Tue Mar 21 06:15:52 2017 -0700  Change lexical analysis print format    Given format for printing out each tokens is as follows.  line number / token / lexeme  Changed TraceScan value as True, which is macro parameter for  main() function. Also, main() function now prints out the as the  given format, changing existing compile logs.  commit 751431350cc62a325b5c714a6b4a469228fd9ad4  Author: Kwanghee Choi <juice500ml@gmail.com>  Date: Tue Mar 21 07:14:35 2017 -0700  [minor] Change indentation  commit 5c5bdcb201146e46e4006069cda71db11f9c66a4  Author: Kwanghee Choi <juice500ml@gmail.com>  Date: Wed Mar 22 09:24:54 2017 -0700  Add lexical analysis for C-style comments    Comment type in C- language is C-style multiline comment, which is  like this.  /\* comment \*/  Lexical analyzer now understands those kind of comments and ignore  characters inside comments, and find out comment blocks with wrong  syntax. This is achieved by the "COMMENT" state. So the analyzer  now has two state, "INITIAL" state, which accepts any viable token,  and "COMMENT" state, which ignores all the characters until \*/ is  found; closing the comment block.  commit fc9635dde405c40d9ca7234c4cb8a06b1082753c  Author: Kwanghee Choi <juice500ml@gmail.com>  Date: Wed Mar 22 09:34:27 2017 -0700  [minor] Reindent cminus.l for consistent spacing  commit 081083e3f2c4c3f198ea343f7ff4f2439c8743ac  Author: Kwanghee Choi <juice500ml@gmail.com>  Date: Wed Mar 22 09:44:36 2017 -0700  Catch line number bug which occurs while analyzing C-style comments    Line number was not incremented if multiline comment is indeed  multiline. Adding rule for newline when in "COMMENT" state fixes  this problem.  commit e227ac2268e6814f5cf2c60889fe826797f34e2c  Author: Kwanghee Choi <juice500ml@gmail.com>  Date: Wed Mar 22 09:49:04 2017 -0700  [minor] Reindent cminus.l for consistent spacing  commit 28b23c59791823ea21379fad6ff51427fc570345  Author: Kwanghee Choi <juice500ml@gmail.com>  Date: Sun Mar 26 20:32:36 2017 -0700  [minor] Fix typo in util.c  commit e5fd660f39137012358413f475c376927d5d67e1  Author: Kwanghee Choi <juice500ml@gmail.com>  Date: Sun Mar 26 22:18:59 2017 -0700  Exclude alphanumeric identifier from lexical analysis    Lexical convention of C- includes rules about whitespaces.  White space consists of blanks, newlines, and tabs. White space is  ignored except that it must seperate ID's, NUM's, and keywords.  Also ID's and keywords are only consisted of lower-case and upper-  case alphabets, and NUM's are only consisted of numbers, 0~9.  So tokens like below are not valid tokens.  abc123, 123abc, abc123def, ...  So, we can conclude that if token consists of both alphabets and  numbers, the token is not a valid token. Those cases were not  caught before. For example, "abc123" was understood as "abc" and  "123" seperately.  commit c460bd16a88a09dd853ffc13a502a7ed272a29c7  Author: Kwanghee Choi <juice500ml@gmail.com>  Date: Sun Mar 26 22:37:17 2017 -0700  Exclude exceptions for analyzing C-style multiline comments    C- is a subset of C, so if definitions are ambiguous, it is safe  to follow ANSI C or C90 Standards. In those standards, comment  start token can appear inside comments, like the following.  /\* Not an error /\* \*/  /\* /\* Not an error \*/  But those cases raised an error in lexical analyzer.  Note that nested comments are still not supported, and thus  analyzer raises an error for the following code.  /\* /\* This is an error \*/ \*/ |

**2. 개발 방법**

개발 환경은 cspro9와 cspro10을 사용하여, Ubuntu 16.04.2 LTS 기준으로 작업한다. vi editor을 이용하여 편집을 하며, Indentation은 GNU Standard를 따른다. 주 개발 언어는 C이나, 이번 Phase의 경우 Lexical Specification 소스 역시 작성한다. 컴파일은 gcc 5.4.0 을 통해 하며, Makefile을 이용하여 빌드 및 Testing의 자동화를 한다. git을 이용하여 Version Control을 하며, Github에서 remote branch를 관리한다.

1. **연구 결과**

**1. 합성 내용**



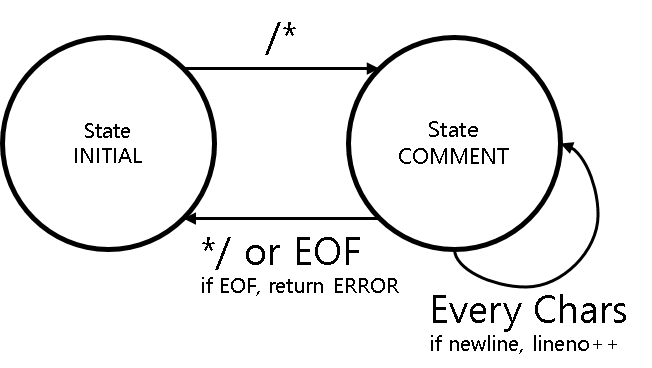
전체 소프트웨어 분석도

* 교과서에서 주어진 TINY Compiler의 소스 코드를 기본으로 설계를 진행했다.
* Build 과정을 자동화하기 위해 Makefile을 사용하여 빌드 Rule들을 지정해주었다.
* 그 과정에서 쓰이는 프로그램이 flex인데, 해당 프로그램은 .lex 파일을 이용하여 각 Token에 대응되는 Rule들을 정해주면, 해당 Rule들을 Parse하여 Hidden Finite State Machine을 생성한다.
* 여기에 추가적으로, lex에 explicit한 state를 설정해줄 수 있는데, C-style multiline comment를 깔끔하게 처리해주기 위하여 “COMMENT” State를 추가해주었다. 이에 대한 구체적인 내용은 2. 분석 내용에서 다루도록 하겠다.

**2. 분석 내용**

교과서에서 주어진 코드는 TINY Compiler으로, 해당 코드를 필요한 만큼 수정하여 사용하였다. Wrapper의 경우, 모든 Compiler가 그렇듯 기본적인 설계는 거의 동일하나 TINY와 C-의 문법이 달라, 해당 문법에 특화된 부분 위주로 수정했고, 제시해준 출력 기준을 만족시키기 위해 출력 형태에도 수정을 가했다. 구체적으로 구현한 각 Corner Case에 대해 살펴보면 다음과 같다.

(1) Multiline Comment 처리



Multiline Comment 처리를 위한 State Machine

COMMENT State는 Comment block start symbol인 /\* 을 만나면 시작되며, \*/을 만나면 종료된다. 하지만 그 전에 EOF를 만나면 제대로 닫히지 않은 Case이므로 ERROR을 Return한다. 해당 방식으로 처리할 경우, 내부에서 각 character을 읽어서 처리하지 않아도 되는 장점이 있다.

(2) Alphanumeric ID 처리

Commit e5fd660의 내용을 발췌하도록 하겠다.

|  |
| --- |
| commit e5fd660f39137012358413f475c376927d5d67e1  Author: Kwanghee Choi <juice500ml@gmail.com>  Date: Sun Mar 26 22:18:59 2017 -0700  Exclude alphanumeric identifier from lexical analysis    Lexical convention of C- includes rules about whitespaces.  White space consists of blanks, newlines, and tabs. White space is  ignored except that it must seperate ID's, NUM's, and keywords.  Also ID's and keywords are only consisted of lower-case and upper-  case alphabets, and NUM's are only consisted of numbers, 0~9.  So tokens like below are not valid tokens.  abc123, 123abc, abc123def, ...  So, we can conclude that if token consists of both alphabets and  numbers, the token is not a valid token. Those cases were not  caught before. For example, "abc123" was understood as "abc" and  "123" seperately. |

해당 근거에 따라, Alphabet과 Number 각 한 Character 이상 있는 Token의 경우, Invalid token이기 때문에 ERROR을 return해주었다.

(3) Multicharacter / Fixed character Tokens

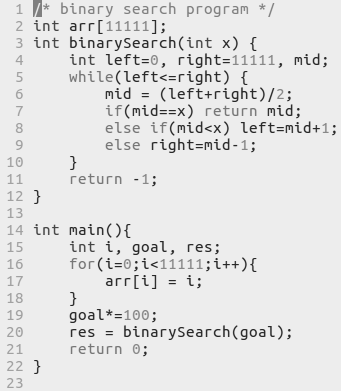
Multicharacter token의 경우 (2)에서 잠시 언급한 바와 같이, ID와 NUM이 있는데 ID는 오직 Alphabet, NUM은 오직 Number만으로 이루어져있으므로 (2)의 경우가 아닌 경우에 ID와 NUM을 구별할 수 있다. 하지만 이 처리는 Fixed character tokens 처리 후에 해야 하는데, 예를 들면 return의 경우 ID로 분류하는 것이 아니라 RETURN으로 분류해야 한다. 그런데 lex의 경우 위에 있을 수록 먼저 선택되므로 모든 Fixed character token을 먼저 정의해야 한다. Keyword는 총 6개, else / if / int / return / void / while 이 존재한다. 특수 Symbol들의 경우 + - \* / < <= > >= == != = , . ( ) [ ] { } /\* \*/ 가 있다.

**3. 제작 내용**

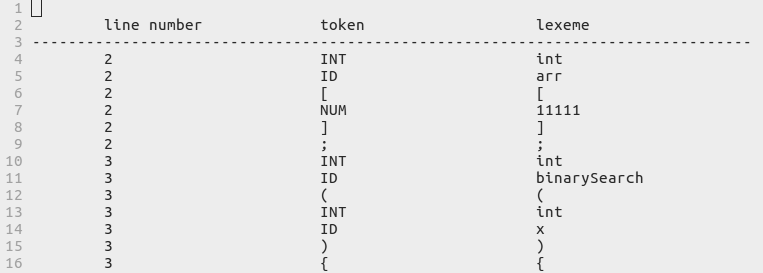
전체 C- Compiler 설계 프로젝트 중 Phase I: Lexical Analyzing of C- Language로, C- Language로 쓰인 소스 코드를 Lexical Analyze를 통해 Token 단위로 쪼갠 뒤 해당 Token과 String을 format에 맞추어 출력할 수 있도록 했다.

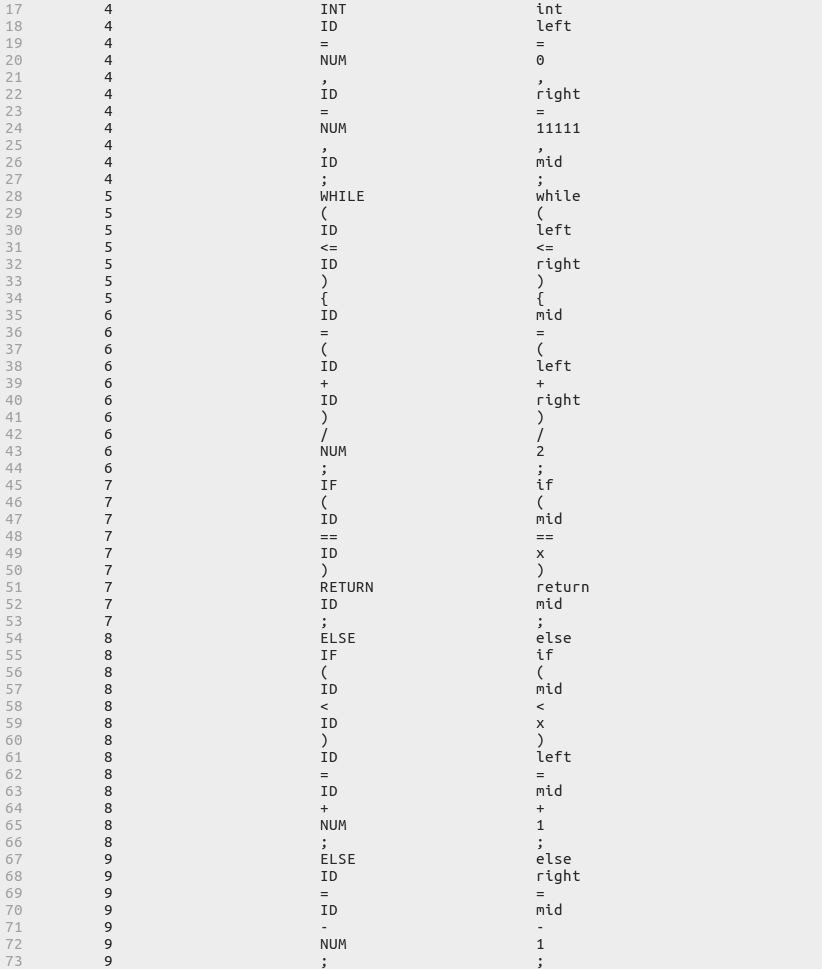
**4. 시험 내용**

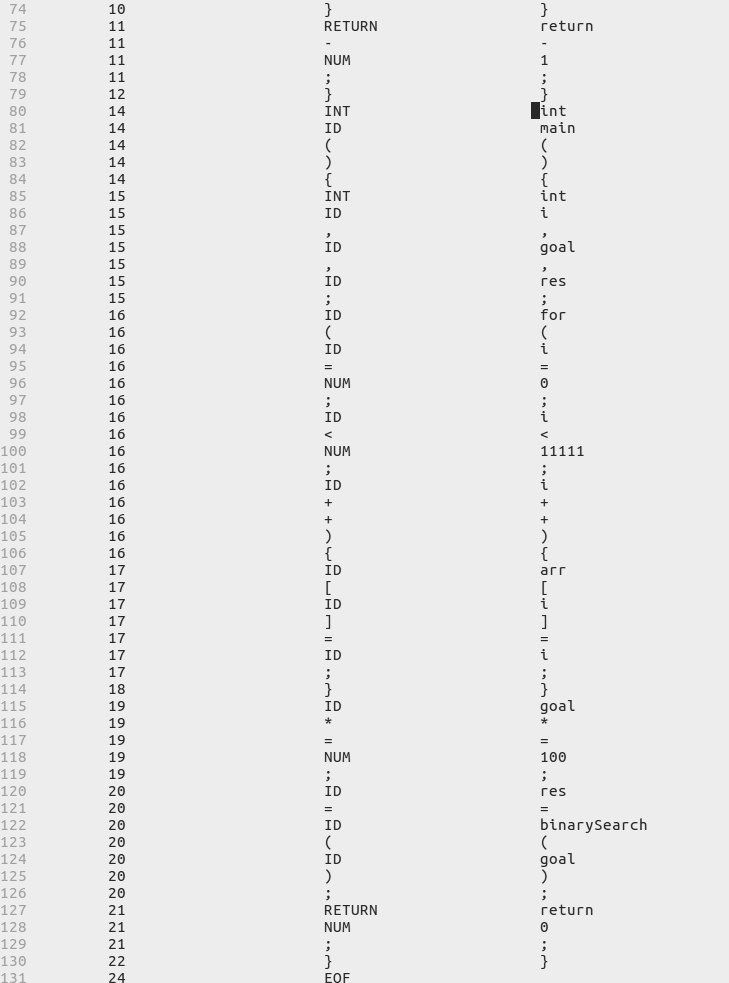
Test #1: 주어진 Test case인 Binary Search (test1.c)



Output #1: 각 Token이 정상적으로 출력됨을 알 수 있다.







Test #2-1: 주어진 Test case인 Comment Corner Case



Test #2-2: 주어진 Test case인 Comment Corner Case

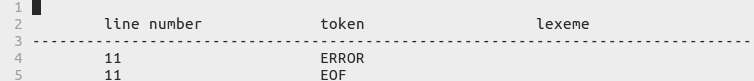


Test #2-3: 주어진 Test case인 Comment Corner Case

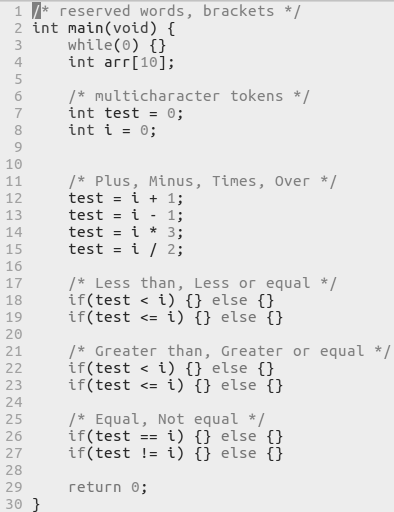


Output #2: 2-1, 2-2, 2-3 모든 Case에 대해 동일한 Output을 갖는다.

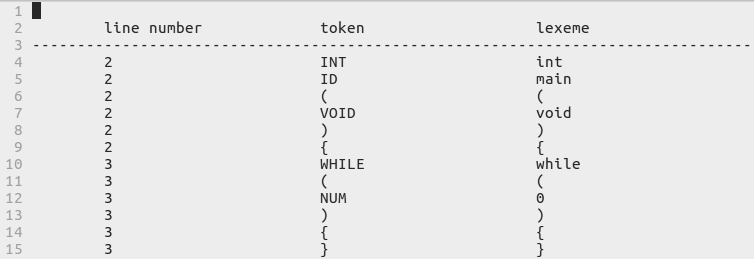
Comment가 닫히지 않았으므로 Error을 Return한다.

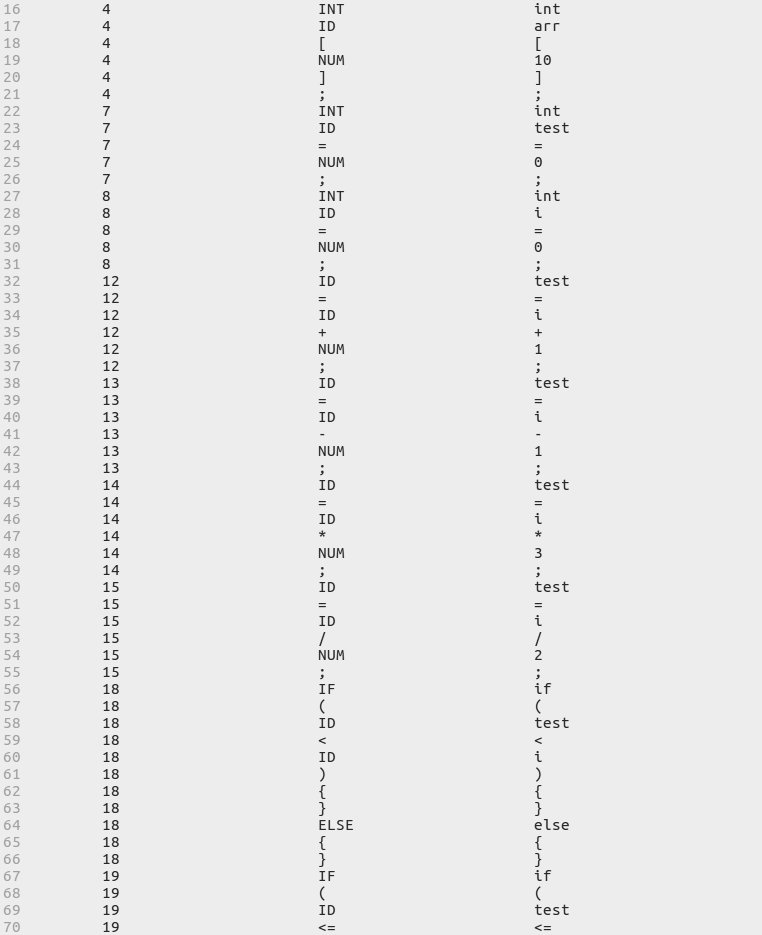


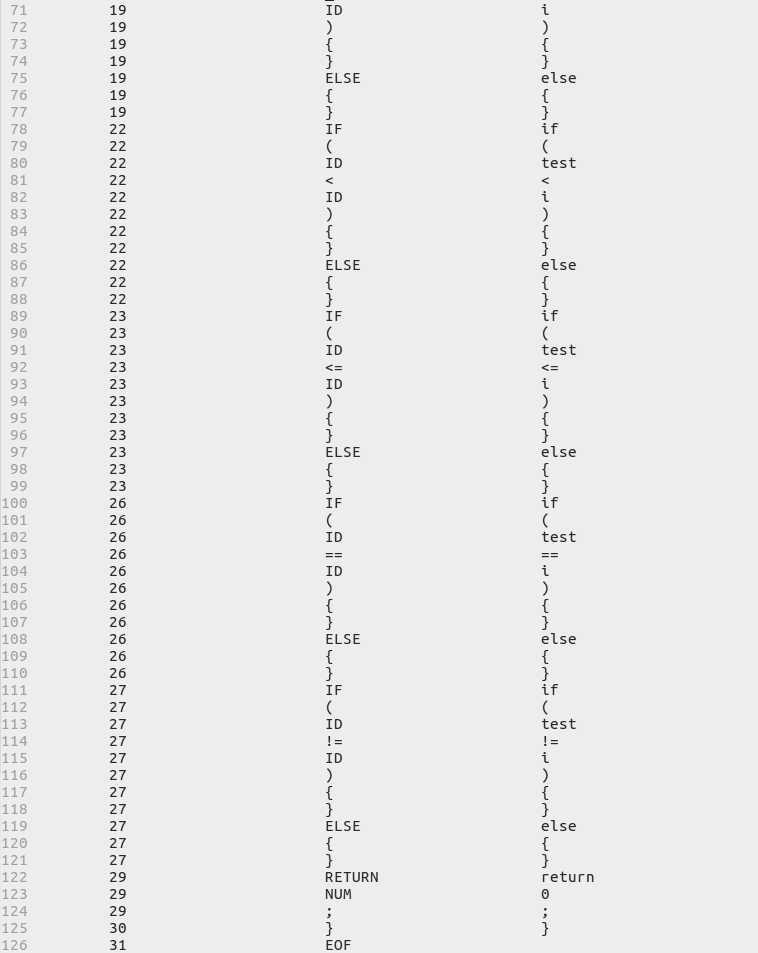
Test #3: C-언어 내의 모든 Reserved token에 대한 검사 (test2.c)



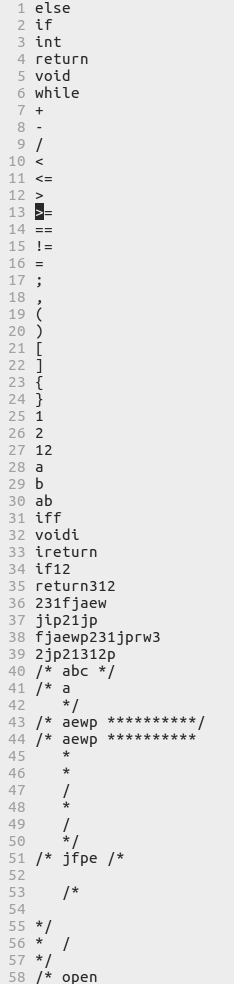
Output #3: 모든 token이 정상적으로 출력됨을 알 수 있다.







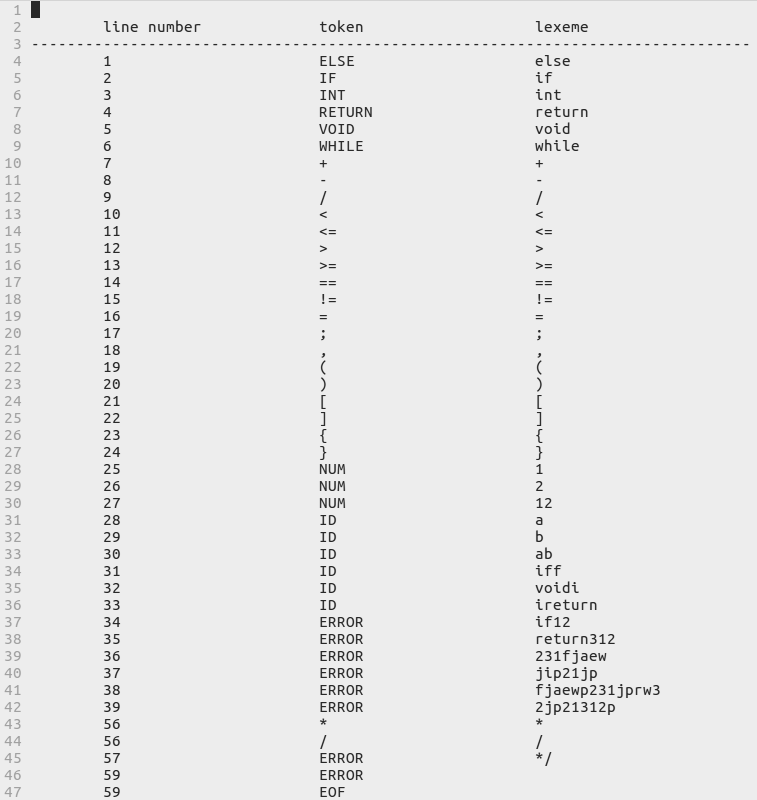
Test #4: Comment 및 Alphanumeric Identifier 예외에 대한 검사

:

Output #4: 정상적인 Token의 경우 잘 출력됨을 알 수 있고,

Line 34~39와 같은 잘못된 Identifier의 경우 ERROR이 나온다.

또한 Line 57, 58와 같이 짝이 맞지 않는 Comment의 경우 ERROR이 나온다.



**5. 평가**

C- 언어의 Syntax가 명확하지 않은 부분이 있어 다소 혼선이 있었으나, Appendix와 C 표준을 참고하여 여러 예외에 대한 해결을 할 수 있었다. 특히, “return123”과 같이 알파벳과 숫자가 섞인 token을 “return”과 “123” 두 개의 token으로 보아야 하는가, 아니면 “return123”이라는 token으로 보고 ERROR을 return해야 하는가에 대한 혼선 등, 여러 문제가 있었으나 기존에 존재하는 규칙들에 어긋나지 않을 수 있도록 고민하여 design 결정을 했고, 그 과정에서 여러 예외에 대해 안정적으로 처리할 수 있는 Lexical Analyzer을 설계할 수 있었다.

개발시에 Git을 사용하여 Commit을 잘게 쪼개 각 변경마다 어떤 사항이 변경되었는가를 Tracking 하도록 하여, 신뢰성 있는 Code 작성 및 Continuous Integration 뿐만 아니라 과제를 제출한 뒤에 지속적으로 타인들과 함께 Open Source Contribution을 할 수 있도록 했다. 또한 문제가 생겼을 때 Revert나 Cherry-pick 등 여러 Version Control 기능을 사용할 수 있어, 개발을 훨씬 안정적으로 할 수 있었다. 해당 코드를 Github에 올림으로써, Local Repository에서만 관리하는 것이 아닌 Remote Repository를 두어, 한 쪽 Code가 날아가더라도 전혀 지장이 없도록 했다.

1. **기타**

위의 5. 평가에서 언급된 바와 같이, C- 언어의 Syntax가 명확하지 않은 부분이 있어, 기존에 존재하는 textbook과 C Language family가 주로 갖게 되는 규칙들에 어긋나지 않도록 design 결정을 했고, 그 과정에서 여러 예외에 대해 안정적으로 처리할 수 있는 Lexical Analyzer을 설계할 수 있었다. 또한, 개발시에 Git을 사용하여 신뢰성 있는 Continuous Integration 뿐만 아니라 이후에도 지속적으로 Open Source Contribution을 할 수 있도록 했다. 문제가 생기더라도 여러 Version Control 기능을 통해 안정적인 개발을 할 수 있었다. 해당 코드는 Github를 통해 Remote Repository를 두어 한 쪽 Repo에 문제가 생기더라도 문제 없도록 했다.

다만, clang이 서버에 깔려있지 않아 Vim package인 clang\_autocomplete package를 사용할 때 직접 clang을 build해서 써야 하는 불편함이 있었다.