**Design and Development of Compiler**

**for C- Language**

Phase 1: Design and Implementation of Lexical Analyze

Results Report of Phase 1

과목명: [CSE4120] 기초 컴파일러 구성

담당교수: 서강대학교 컴퓨터공학과 정 성 원

개발자: 20141522 박상욱

개발기간: 2017. 3.20 - 2017. 3. 27

각 단계별 결과 보 고 서

프로젝트 제목: Design and Development of Compiler for C-Language:  
Phase 1: Design and Implementation of Lexical Analyzer

**제출일: 2017.** **3.** **27.**

**개발자: 20141522박 상 욱**

**I. 개발 목표**

본 프로젝트는 C-(C MINUS) Compiler를 구현해봄으로써, Compiler를 이해하는 것을 목표로 한다. Phase 1 에서는 Compiler Structure의 기초인 Lexical Analyzer(Scanner)를 구현한다. Lexical Analyzer는 C- 언어로 작성된 c 파일을 입력 받아, Token과 Lexeme를 분석하여 결과를 보여주는 프로그램이다. Output으로 Input 파일을 Line number에 해당하는 숫자와 Token의 이름, Lexeme를 line by line으로 출력한다.

II. 개발 범위 및 내용

1. 개발 범위

Phase 1의 개발 목표는 C- Language의 Lexical Analyzer 개발이다. Phase 1의 최종 프로그램은Regular Expression을 사용해서 C-로 작성된 코드를 분석하여 각 Token에 해당하는 Lexeme를 찾을 수 있도록 한다. 이를 위해 주어진 코드[[1]](#footnote-1)를 이용하여 Lexical Analyzer를 개발하며, FLEX 툴을 이용하여 완성한다.

1. 개발 내용

본 프로젝트에서는 C 언어가 아닌, C 언어에서 일부를 추출한 C- 문법을 이용해서 개발한다. C- 문법에 해당하는 Reserved word를 추가하며, ‘/\* \*/’로 나타낼 수 있는 주석에 대한 문법을 추가한다. 이외에도 추가적인 Special Symbol을 C-문법에서 사용할 수 있도록 한다. Lexical Analyzer로 C- 코드를 분석하는 경우, Line Number, Token, Lexeme 순으로 분석된 결과를 출력할 수 있도록 개발한다.

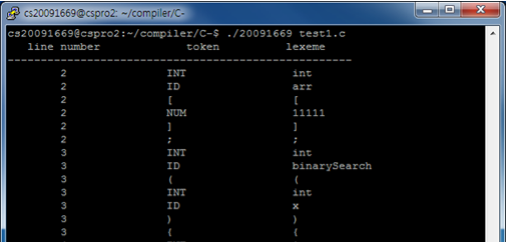


Figure . 프로젝트 목표 결과물

**III. 추진 일정 및 개발 방법**

1. **추진 일정**

* **3.20 ~ 3.22 :** 프로그램 요구사항 분석
* **3.22 ~ 3.23 :** 주어진 프로그램 코드 분석
* **3.23 ~ 3.25 :** FLEX 툴을 이용해 Lexical Analyzer 제작
* **3.25 ~ 3.26 :** 프로그램 Test
* **3.26 ~ 3.27 :** 프로젝트 결과보고서 작성

1. **개발 방법**

OS X에서 개발을 진행하며, brew를 이용해 FLEX 툴을 설치[[2]](#footnote-2)해 진행한다. Lexical Analyzer를 제작하는 데에 있어 FLEX툴을 이용해 Regular Expression으로 Tokenize를 진행한다. C- Language 문법을 사용하며, 다운받은 기존의 TINY Compiler 코드(main.c, globals.h, util.c, tiny.l 등)을 수정하여 프로그램을 개발한다. 추가적인 Token과 Comment Error에 대한 사항을 작성한다.

**Ⅳ. 연구 결과**

1. **합성 내용**

본 프로젝트에서는 C- Language를 정의[[3]](#footnote-3)하고 사용한다. 따라서 프로젝트를 통해 개발되는 프로그램은 C- Language에 대한 Lexical Analyzer 기능을 한다. Lexical Analyzer 개발을 위해 Tokenize 과정을 Regular Expression을 이용한다. 이 과정을 FLEX 툴을 이용하여 개발한다. FLEX는 Tokenizer로 Text에서 Regular Expression을 이용해 Lexical Pattern을 찾는 툴이다. 본 프로젝트에서 FLEX는 tiny.l 파일에 정의된 Regluar Expression으로부터 lex.yy.c 파일을 생성하며 다른 main.c util.c 프로그램과 함께 컴파일 하여 최종 Lexical Analyzer 프로그램을 만든다.

Lexical Analyzer를 제작하기 위해서는 앞선 프로그램들을 컴파일한다. make명령을 통해서 컴파일이 가능하다. 작성된 프로그램은 20141522 라는 이름의 실행프로그램이다. 프로그램의 실행은 ./20141522 [Filename] 과 같은 방법을 통해 가능하다. 프로그램의 결과는 <Figure 1>과 같이 입력받은 File에 대한 Line number와 Token 그리고 Token에 해당하는 Lexeme가 출력된다.

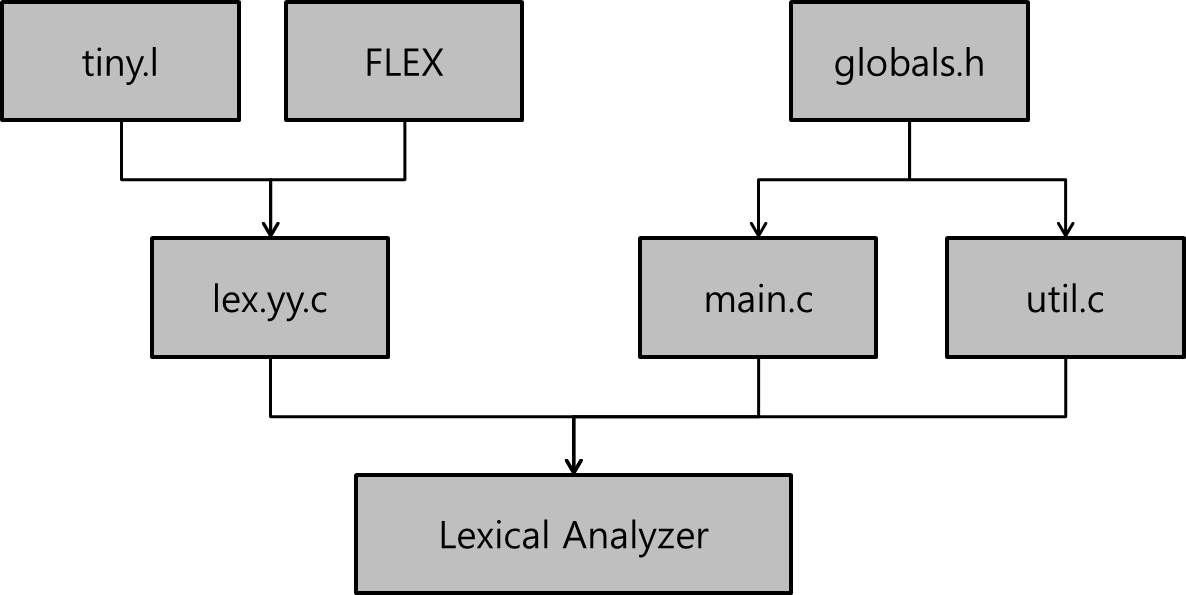


Figure . 프로그램 흐름도

1. **분석 내용**
   1. **main.c**

Lexical Analyzer생성에 있어서 기본이 되는 소스이다. Phase 1에서는 Scanner인 Lexical Analyzer제작만을 목표로 하므로. 이에 해당하는 NO\_PARSE를 TRUE로 설정해준다.

* 1. **globals.h**

Lexical Analyzer 생성 코드에서 사용될 define, 전역 변수(lineno, listing), Token 이름을 정의한다.

* 1. **util.c**

Lexical Analyzer에서 필요한 함수는 printToken 함수이다. 입력받은 파일의 Lexeme에 해당하는 Token을 찾고 양식에 맞추어 출력한다.

* 1. **tiny.l**

Regular Expression을 이용해 C- Language에서 사용한 문법을 기록한다. 특히 사용되는 문법을 정리하면,. 아래와 같다.

Keyword : if, else, int, return, void, while

ID:letter|letter\*(letter=a|..|z|A|..|Z)

NUM : digit|digit\*(digit=0|..|9)

Special Symbol :+ - \* / < <= > >= == != = ; , ( ) [ ] { } /\* \*/

1. **제작 내용**
   1. **main.c**

Lexical Analyzer를 컴파일하기 위해서, NO\_PARSE를 TRUE로 값을 변경한다. Scanner를 통해 분석한 결과를 정리해 보여주기 위해 상단의 열 제목을 추가한다

Lineno Token Lexeme

--------------------------------------------------------------------------------------------

* 1. **globals.h**

C- 문법에 해당하는 예약어를 추가하였다. 추가된 예약어는 INT, WHILE, RETURN, VOID이다. 또한 ‘/\* \*/’로 나타낼 수 있는 Comment에 대한 Token을 추가하였다. Special Symbol에 대해서도 tiny.l에 나타난 모든 symbol에 대한 Token을 추가하였다.

* 1. **util.c**

printToken함수에서 global.h에 추가한 Token에 대해 Case를 추가하였다. 단순히 Lexical Analyzer는 Scanner의 역할을 하기에 Lexeme와 그에 해당하는 Token을 출력하게끔 작성하였다.

* 1. **tiny.l**

분석 내용에서 설명한 추가적인 Token에 대해 Regular Expression을 이용해 추가하였다. Comment에 대한 부분은 아래와 같이 처리하였다.

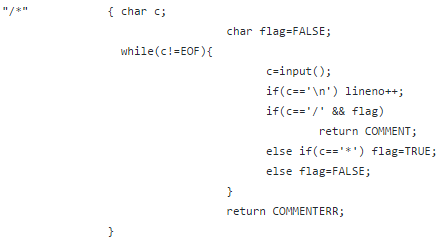


Figure . Comment Token에 대한 처리 코드

위와 같이 ‘/\*’ Lexeme이후로 ‘\*’문자가 나오는지 검사하며, ‘\*’문자와 ‘/.’문자가 연속으로 나오는 경우에만 Comment Token으로 인식하도록 처리하였다. 그렇지 않은 경우에는 파일의 끝까지 탐색하여 Comment Error가 나도록 하였다.

getToken함수에서 Line Number를 출력하도록 수정하였고, 현재 토큰이 Comment인 경우에는 그 내용을 출력하지 않도록 수정하였다.

1. **시험**

본 프로젝트, Lexical Analyzer를 시험하기 위해서 C- Language를 따르는 Test File을 제작하였으며, 그 File은 아래와 같다.

* 1. test1.tny

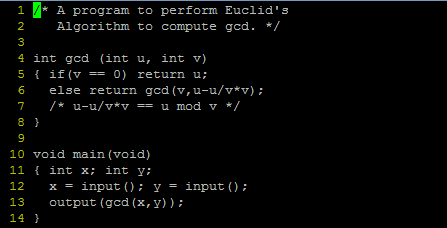
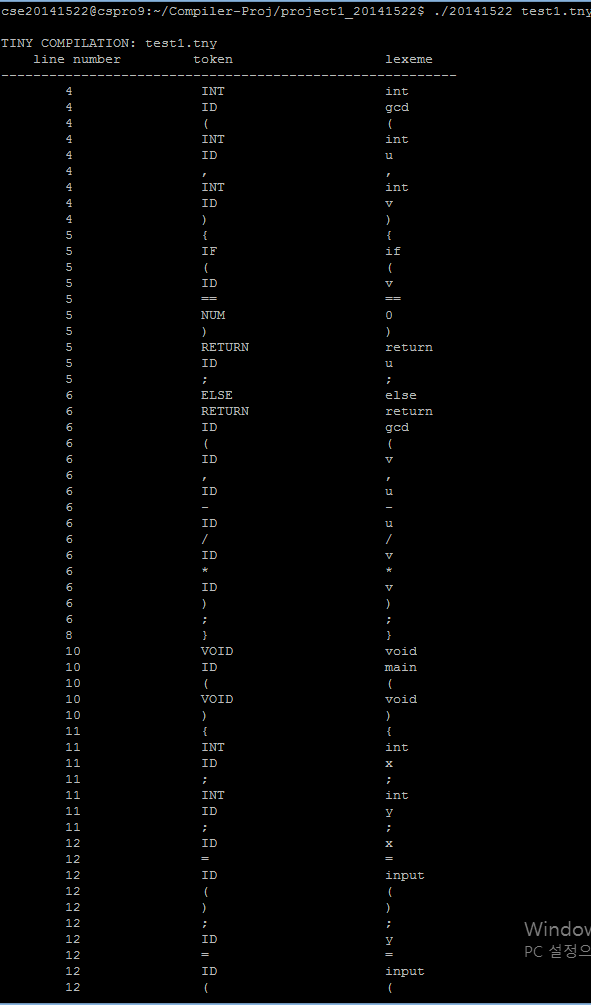
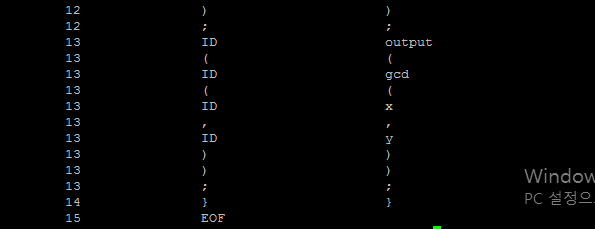


Figure . test1.tny 코드





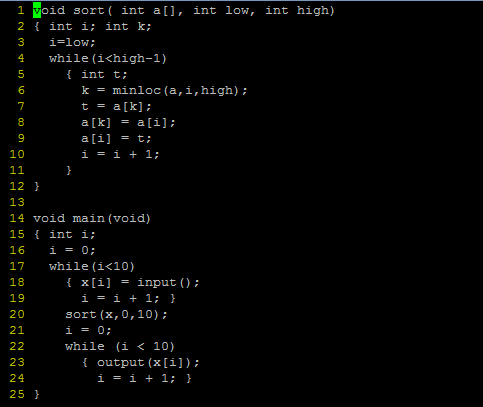
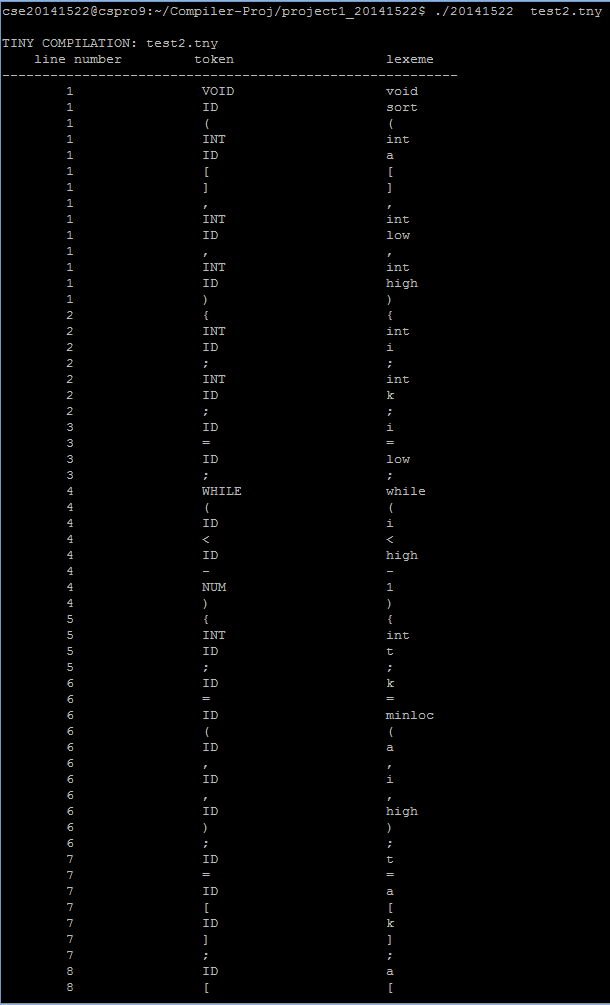
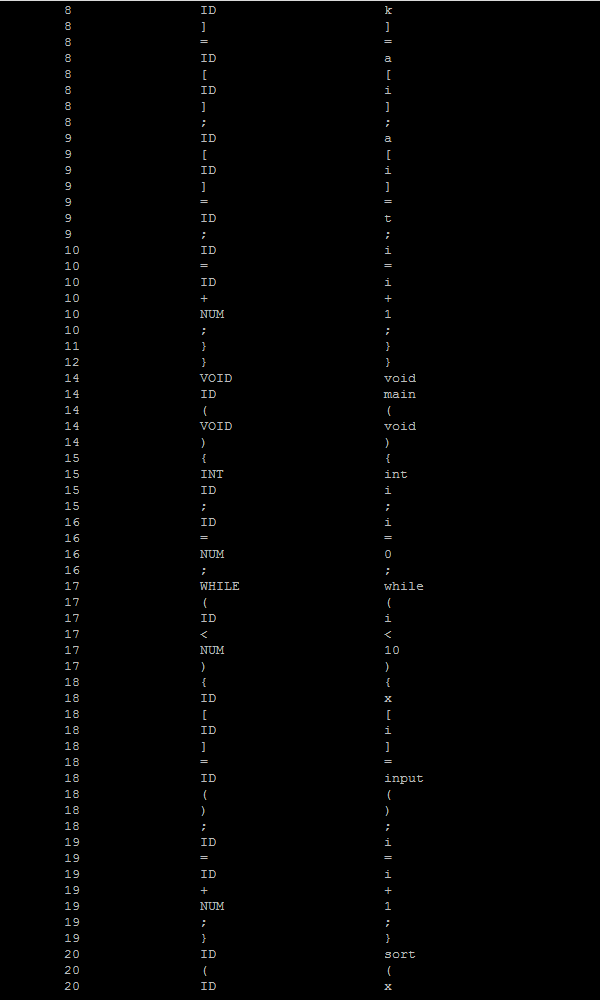
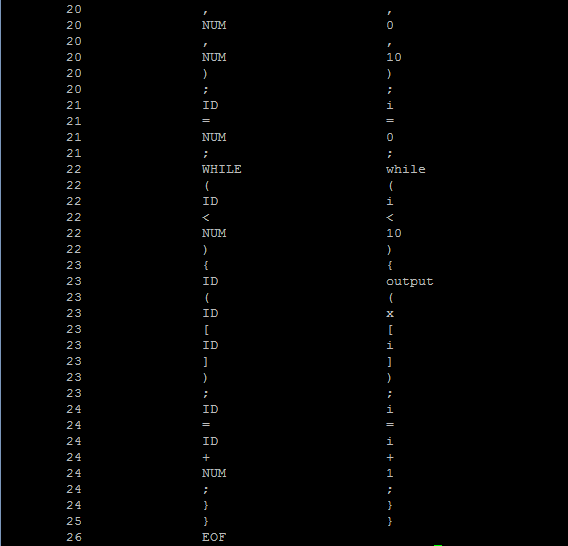
* 1. test2.tny

Figure . test2.tny 코드



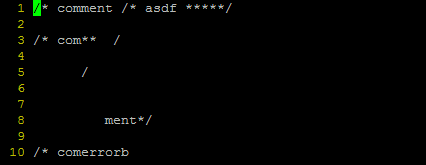
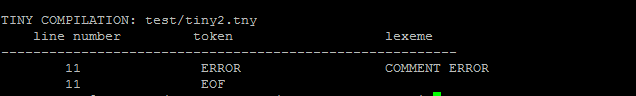
* 1. Comment Error

Figure . Comment Error 코드



1. **평가 내용**

본 프로젝트는 Appendix A의 C- Language 문법을 정확히 따라 제작되었다. 따라서 앞서 Test한 C- Language로 작성된 프로그램들을 정확히 분석하는 것을 확인 할 수 있었다. Error Token에 대해서, Error임을 명시해 주었고, Comment Error에 대해서도 EOF까지 탐색하여 정확히 찾을 수 있도록 처리하였다. 또한 동일한 Test File에 대해서 반복적 수행을 통해서도 프로그램 수행에 있어 문제가 발생하지 않았다. Mac OS X 10.12.3과 Ubuntu 16.04.2에서 Test를 모두 진행했음에도 문제가 발생하지 않았다.

**V. 기타**

1. **자체 평가**

제작한 Lexical Analyzer는 의도한 C- Language에 대해서 훌륭한 Scanner 역할을 한다. 시험을 위해 제작했던 Test File들을 문제없이 통과하였다. 프로젝트를 진행하는 과정에 있어 GitHub를 사용해 코드를 관리하여, 신뢰성을 높였다.

1. **느낀 점**

Lexical Analyzer를 제작하게 되면서, Lexeme, Token에 대해 좀 더 자세히 알 수 있었다. C언어 파일에서 FLEX 툴을 이용해 간편히 Tokenize가 가능하다는 것을 배울 수 있었다. 평소 프로그래밍 작업을 하면서 Compiler가 어떻게 식별자를 구분하고, 오류를 잡아내었던 것인지 조금은 이해할 수 있었다.

1. <http://www.cs.sjsu.edu/~louden/cmptext/> 에서 자료를 다운받을 수 있다. [↑](#footnote-ref-1)
2. brew install flex 명령을 이용해 FLEX 설치가 가능하다. [↑](#footnote-ref-2)
3. C- Language에 대한 정의는 Kenneth C. Louden의 Compiler Construction Principles and Practice, Appendix A(p.491~501)를 참조한다. [↑](#footnote-ref-3)