**Design and Development of Compiler for C- Language**

**Final Report (결과 보고서)**

과목명: [CSE4120] 기초 컴파일러 구성

담당교수: 서강대학교 컴퓨터공학과 정 성 원

개발자: 권태국

개발기간: 2017. 3. 20 - 2017. 3. 27

**프로젝트 제목: Design and Development of Compiler for C-Language:**

**Phase 1: Design and Implementation of Lexical Analyzer**

**제출일: 2017. 03. 27.**

**개발자: 권태국**

# I. 개발 목표

이번 프로젝트에서는 C- language에 대한 어휘 분석기 (Lexical Analyzer) 를 설계하고 개발한다.

# II. 개발 범위 및 내용

## 가. 개발 범위

C- language에 대한 어휘 분석기를 설계하고 개발한다.

## 나. 개발 내용

C- language의 어휘 분석을 위해서 일단 token 종류들에 대해서 각각 정규표현식을 작성한다. 그리고 그 정규표현식들을 바탕으로 C- lex file을 작성한다. 그 후 flex와 C언어를 이용해서 어휘 분석기 프로그램을 작성한다. 그리고 이를 테스트하기 위해 C-문법을 이용하여 몇 가지 예제 프로그램을 작성하여 개발된 어휘 분석기를 테스트한다.

# III. 추진 일정 및 개발 방법

## 가. 추진 일정

3월 20일 : proposal 작성.

3월 21일 ~ 3월 22일 : 정규표현식 작성.

3월 23일 : 정규 표현식을 바탕으로 lex file 작성, C- 예제 프로그램 작성.

3월 24일 ~ 3월 25일 : C언어를 이용해 lexical analyzer 설계 및 개발.

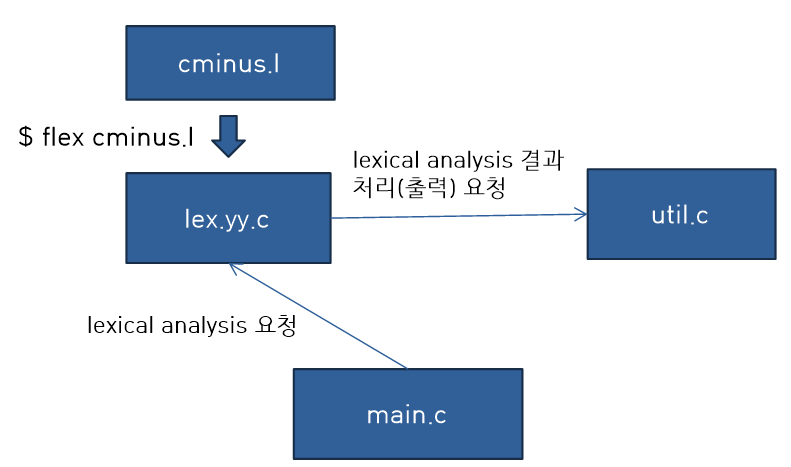
3월 26일 : 결과보고서 작성.

## 나. 개발 방법

C- 언어의 키워드등의 명세를 확인한다. 그 후 어휘 분석에서 분류될 token들의 리스트를 작성한 후, 각 token들을 인식하기 위한 오토마타를 생성하기 위해 정규표현식들을 작성한다. 그리고 그 정규표현식들을 바탕으로 lex file을 작성한다. 여기서 /\* \*/ 주석을 인식하기 위해서 lex 문법중 %% "/\*" { … } 를 통해서 /\* \*/ 를 인식하기 위한 코드를 작성한다.

# IV. 연구 결과

## 가. 합성 내용



전체 소프트웨어의 구성도는 위와 같다. cminus.l은 c- language의 lexical analysis 생성을 위한 lex 파일이다. flex를 통해 cminus.l 를 가지고 lex.yy.c 를 생성할 수 있다. lex.yy.c는 lexical analysis에 관련된 기능을 하는 모듈이다. util.c는 lexical analysis의 결과를 처리(결과 출력)하는 등 소프트웨어의 유용한 util들을 담고있는 모듈이다. main.c는 메인 모듈로서 source file을 읽고, 관련 전역변수들을 초기화하는 등의 작업을 수행한다.

cminus.l 에는 각 어휘들을 인식하기 위한 정규표현식, 추가적인 코드들을 들어있다. flex는 이 lex파일내에 정의되어있는 정규표현식을 가지고 NFA를 만들고, NFA를 DFA로 변환하는 알고리즘을 이용해 NFA를 DFA로 변환한다. 우리는 런타임에 이렇게 만들어진 DFA를 이용해 어휘들을 인식하게 된다. 주석(/\* \*/)같은 경우에는 오토마타를 이용하여 처리하기가 어렵다. 따라서 따로 처리 코드를 작성하여 처리하게 된다.

사용자가 만들어진 소프트웨어를 사용하기 위해서는 make 명령어를 이용해 빌드한 후, 만들어진 실행파일에 파라미터로 C- source file의 path를 넣어주면 된다.

## 나. 분석 내용

cminus.l 에 각 어휘들을 인식하기 위한 정규표현식과, 주석을 처리하기 위한 추가적인 코드를 작성하였다. 구체적인 내용은 cminus.l 파일을 참고하면 된다. 주목할만한 정규표현식은 아래와 같다.

{letter}+{number}({letter}|{number})+

{number}+{letter}({letter}|{number})+

이 두개의 정규표현식에 해당하는 어휘는 ERROR로서 처리된다.

이 두개의 정규표현식이 인식하는 어휘의 예시는 아래와 같다.

* hello12
* return12
* 123hello
* 12hhhh44

이러한 어휘들은 따로 정규표현식을 작성하여 인식해 ERROR로서 처리해줘야한다.

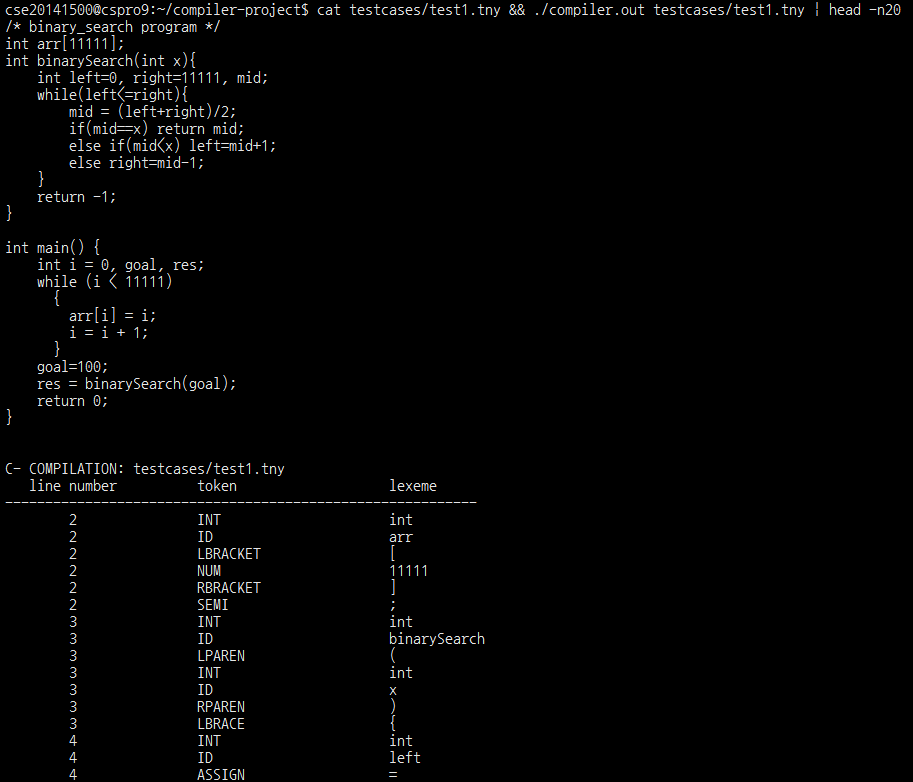
그렇지않으면 return12 같은 것들이 return 과 12로 나눠서 인식되어 정상 처리되게 된다.

따라서 나는 두 개의 정규표현식을 추가하여 이러한 예외케이스들을 처리하였다.

lex.yy.c는 flex에 cminus.l를 넣으면 자동 생성된다. util.c에는 lex.yy.c에서 인식된 어휘들을 화면에 출력해주는 역할을 한다. main.c는 전체 소프트웨어를 초기화하고 lex.yy.c를 이용해 lexical analysis를 하는 역할을 한다.

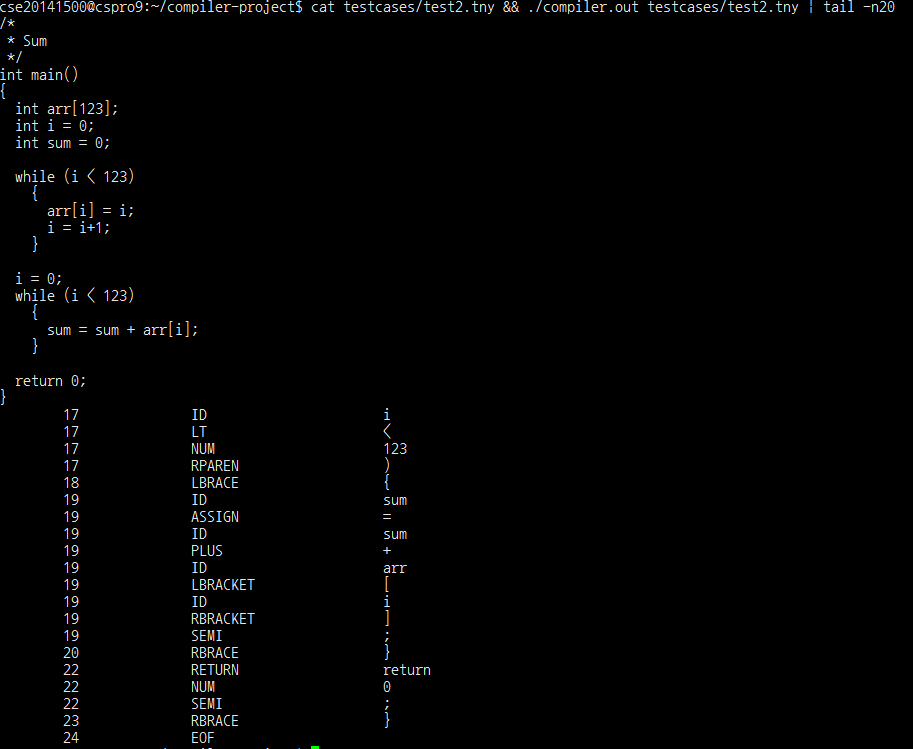
## 다. 제작 내용

### 1. C- language로 작성된 binary search program (test1.tny) 어휘 분석 결과



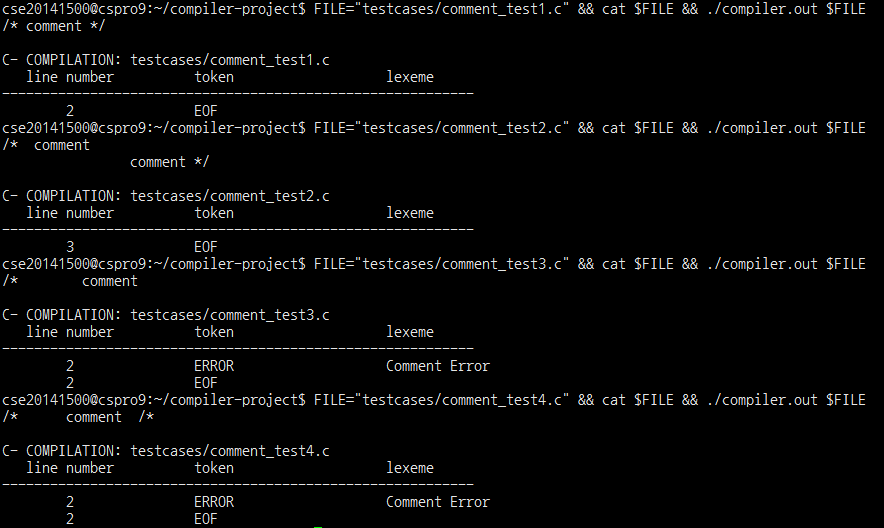
분석 결과가 매우 길어서 앞의 20줄만 첨부하였다.

### 2. C- language로 작성된 sum program (test2.tny) 어휘 분석 결과



분석 결과가 매우 길어서 뒤에 20줄만 첨부하였다.

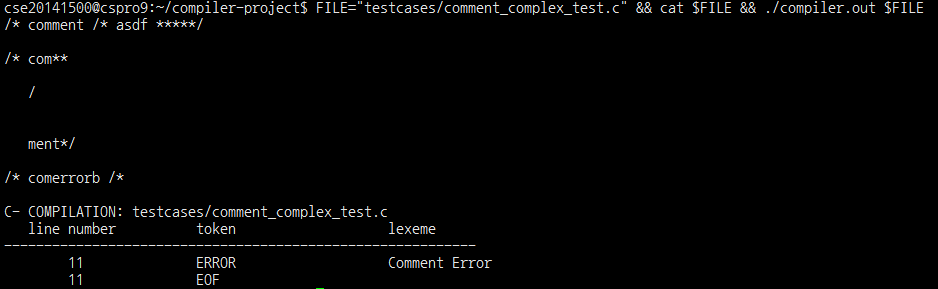
### 3. Comment 처리 테스트 (4가지 테스트 케이스)



프로젝트 설명서에서 요구한 주석관련 4가지 케이스에 대한 주석 처리 테스트이다.

설명서에 나온 것과 똑같이 동작하는 것을 알 수 있다.

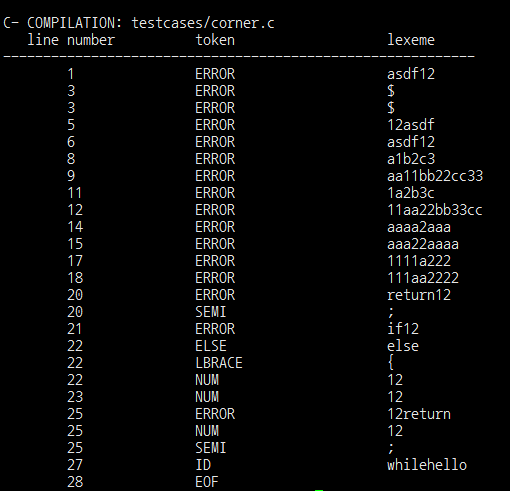
### 4. Complex Comment 처리 테스트



조교님이 첨부한 example pdf에 나와있는 주석관련 테스트 케이스에 대한 결과이다. 정확히 동작하는 것을 볼 수 있다.

### 5. Corner case 처리 테스트





위는 corner case에 대한 테스트케이스와 그에 대한 lexical analysis 결과이다. 각종 corner case에 대해서 정확히 동작하는 것을 알 수 있고, 정규표현식들과 lex file이 잘 작성됐음을 알 수 있다.

## 라. 시험 내용

이번 프로젝트에서 제작한 C- language lexical analysis 소프트웨어의 평가 기준은 아래와 같다.

* 기본적인 Identifier, Number, Keyword, Special Symbol 들을 인식하는 정규표현식을 작성한다.
* 주석 (/\* \*/) 을 처리하기 위한 로직을 작성하여 lex file에 포함시킨다.
* 각종 corner case들을 고려하고, 이를 해결하기 위한 정규표현식등을 lex file에 반영한다.

그리고 나는 위 3가지 모두를 달성하였다.

기본적인 identifier, number, keyword, special symbol들을 인식하는 정규표현식을 작성하고 주석을 처리하기 위한 로직을 작성하여 lex file에 포함시켰다. 그리고 이 두가지를 테스트하기 위한 테스트케이스 (test1.tny , test2.tny , comment\_test1.c , comment\_test2.c , comment\_test3.c , comment\_test4.c , comment\_complex\_test.c) 을 작성하고 테스트하였다.

그리고 또한 나는 corner case에 해당하는 테스트 케이스 (corner.c) 를 작성하고 이 것에 정상적으로 동작하기 위해 추가적으로 error를 검출하는 2가지 정규표현식을 작성하여 lex file에 추가하였다. 이렇게 함으로서 나는 기존의 3가지 기준을 모두 만족시킬 수 있었다.

## 마. 평가 내용

(라)의 시험내용에서 말했듯이 난 3가지 기준을 모두 만족시키는 lexical analysis software를 개발하였다. 내 소프트웨어의 장점은 return12 같은 corner case들에도 잘 작동한다는 것이다. 초기에 개발을 완료했을 때는 corner case에 대한 고려가 안되어있었다. 그러나 test를 진행하는 과정에서 corner case들에 대해 정확히 동작을 하지 않는 것을 발견했다. 그래서 이를 해결하기 위해 정규표현식을 2개 추가하게 되었다. 이러한 테스트과정을 통해서 계속 소프트웨어가 보완됐으므로 안정적이고 신뢰가 있다고 할 수 있다.

# V. 기타

## 가. 자체 평가

나는 추가적으로 corner case들을 작성하여 나의 프로그램을 검증하고, corner case들에 맞게 프로그램이 잘 작동하도록 수정하였기 때문에 더 좋은 점수를 받아야 한다고 생각한다. 그리고 나는 bitbucket을 이용하여 git으로 프로젝트의 형상관리를 수행하였다.

## 나. 느낀점

비록, C언어는 아니지만, C의 subset인 C- language의 lexical analyzer를 만들어보면서 참 재밌고, 신기함을 많이 느꼈다. 그동안, 막연하게 어렵게만 보이던 컴파일러를 개발하는 그 시작점으로서 큰 의미가 있다고 느껴진다. 앞으로 남은 컴파일러 프로젝트도 성실하고 열심히 수행하여, 컴파일러 제작의 기본적인 기술과 원리를 습득하도록 노력할 것이다. 그리고 학기가 끝난 후에는 추후 나중에 나만의 프로그래밍 언어를 개발하고, llvm등을 활용하여 그것의 컴파일러를 제작해볼 생각이다.