**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. 4. 26 - 2017. 5. 12

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

**Phase 2: Design and Implementation of LALR Parser**

**제출일: 2017. 05. 12.**

**개발자: 권태국**

# I. 개발 목표

이번 프로젝트에서는 C- language에 대한 parser (즉, syntactic analyzer)를 만드는 것이 개발 목표이다. Parser의 최종 output은 abstract syntax tree이다.

# II. 개발 범위 및 내용

## 가. 개발 범위

C- language에 대한 parser (즉, syntactic analyzer)를 개발한다. Parser의 최종 output은 abstract syntax tree이다.

## 나. 개발 내용

Project 1에서 개발한 lexical analyzer가 산출하는 token list들을 가지고 abstract syntax tree를 만들어내는 것이 바로 이번 프로젝트에서 개발할 parser이다. 이 parser는 LALR을 기본으로 하며 생기는 conflict는 우선 순위와 associativity를 통해 해결하는 방법을 사용할 것이고, 이를 위해서 Bison을 사용할 것이다. 그리고 결과로서 만들어지는 abstract syntax tree를 적절한 형태로 사용자에게 보여준다.

# III. 추진 일정 및 개발 방법

## 가. 추진 일정

4월 26일 : Proposal 제출

4월 27일 ~ 5월 2일 : C- 언어의 grammar specification 작성

5월 3일 ~ 5월 7일 : Abstract Syntax Tree를 만들 수 있도록 하는 semantic action 작성

5월 8일 ~ 5월 11일 : 기존의 lexical analyzer와 병합 및 종합 테스트

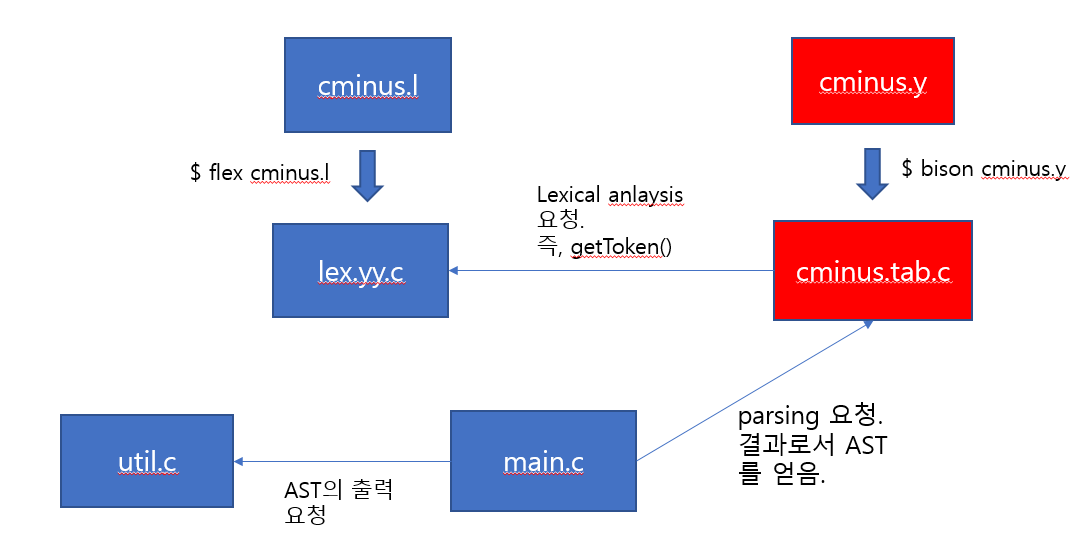
5월 12일 : Result Report 제출

## 나. 개발 방법

C- 언어의 syntax에 대한 명세를 확인한다. 그리고 syntax들을 parsing할 수 있는 grammar의 specification 파일을 작성한다. 이 파일내에 grammar의 specification과 더불어 semantic action들을 통해 Abstract Syntax Tree를 만들 수 있도록 한다. 이렇게 만들어진 specification 파일은 Bison을 통해 compile되고, 이렇게 만들어진 parser는 lexical analyzer와 상호작용하게 된다. 그리고 결과로서 반환되는 abstract syntax tree를 알아볼 수 있는 형태로 사용자에게 보여준다.

# IV. 연구 결과

## 가. 합성 내용



전체 소프트웨어의 구성도는 위와 같다. cminus.l은 c- language의 lexical analyzer생성을 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. Cminus.y는 c- language의 parser를 생성하기 위한 yacc파일이다. Bison을 통해 cminus.y를 가지고 cminus.tab.c를 생성할 수 있다. Cminus.tab.c는 parsing을 해서 결과로서 abstract syntax tree를 산출해내는 모듈이다. util.c는 abstract syntax tree를 사용자에게 출력하는 등 소프트웨어의 유용한 util들을 담고있는 모듈이다. main.c는 메인 모듈로서 source file을 읽고, 관련 전역변수들을 초기화하는 등의 작업을 수행한다.

cminus.y 에는 token list를 바탕으로 abstract syntax tree를 생성하기 위한 syntax rule, semantic action, 추가적인 코드들이 들어있다. Bison은 LALR을 바탕으로 cminus.y에 명시된 syntax를 인식할 수 있는 오토마타와 table들을 생성한다. 그리고 이러한 기능을 수행하는 소스코드들을 생성해준다.

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

## 나. 분석 내용

Cminus.y에는 token list를 바탕으로 abstract syntax tree를 생성하기 위한 syntax rule, semantic action, 추가적인 코드들이 들어있다. 일단 교재의 Appendix A를 참고하여 C- language의 syntax rule을 파악하고 이를 cminus.y에 옮겼다. 그리고 이제 syntax rule에 따른 파싱의 결과로서 abstract syntax tree를 얻기 위해서 관련된 코드를 semantic action 부분에 작성하였다.

특별하게 처리해줘야 했던 점은 3가지가 있었다.

첫 번째로, 내가 프로젝트 1에서 작성한 lexical analyzer는 주석을 발견할 경우 token으로서 COMMENT 를 반환한다. 그런데, parser에서 syntax rule에는 COMMENT에 대한 부분이 없으므로, 소스코드에 주석이 있으면 parsing error가 떴었다. 그래서 나는 이 부분을 해결하기 위해 cminus.y의 yylex 함수에서 token이 COMMENT가 아닐 때 까지, getToken을 반복해서 호출하도록 코드를 수정하였다. 그렇게 해서 성공적으로 이를 해결할 수 있었다.

두 번째로, shift-reduce conflict 의 해결이다. bison으로 parser를 생성할 때 shift-reduce conflict 경고가 나왔었다. 문제는 ELSE 에 관련된 부분이었다. 나는 이를 해결하기 위해 RPAREN과 ELSE의 우선순위를 조절하였다. 이를 위해서

%nonassoc RPAREN

%nonassoc ELSE

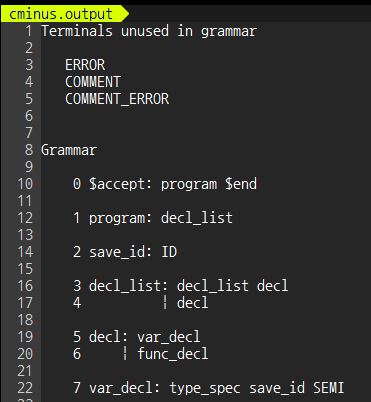
와 같은 부분을 cminus.y 에 넣어주었다. 이렇게 함으로서 해결을 할 수 있었다.

세 번째로, tokenString이 자꾸 다음 token을 가리키는 문제점이다. Syntax rule이 있을 때, rule의 중간부분에서 semantic action을 통해 tokenString을 가지고 tree node에 어떤 정보를 기록해야 하는 경우가 있다. 예를 들면, 함수 선언 룰의 경우, 함수의 이름으로써 tokenString을 넣는 등이 있다. 근데, 자꾸 나는 이상한 값이 tree node로 들어갔다. 알고보니, parsing과정에서 syntax rule의 다음 token에 해당하는 tokenString이 미리 참조되어, 문제가 발생했다. 따라서 나는 이 문제를 해결하기 위해, tokenString에 접근하는 부분 자체를 일종의 syntax rule로 따로 만듬으로써 이러한 문제를 해결했다.

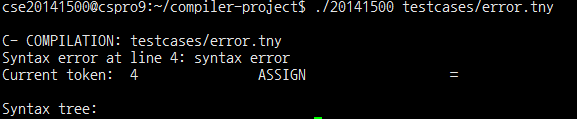
util.c에는 cminus.tab.c에서 생성된 abstract syntax tree를 화면에 출력해주는 역할을 한다. main.c는 전체 소프트웨어를 초기화하고 cminus.tab.c를 이용해 parsing을 하는 역할을 한다.

## 다. 제작 내용

### C- language에 대한 yacc 파일의 conflict 없음을 증명하는 사진 자료

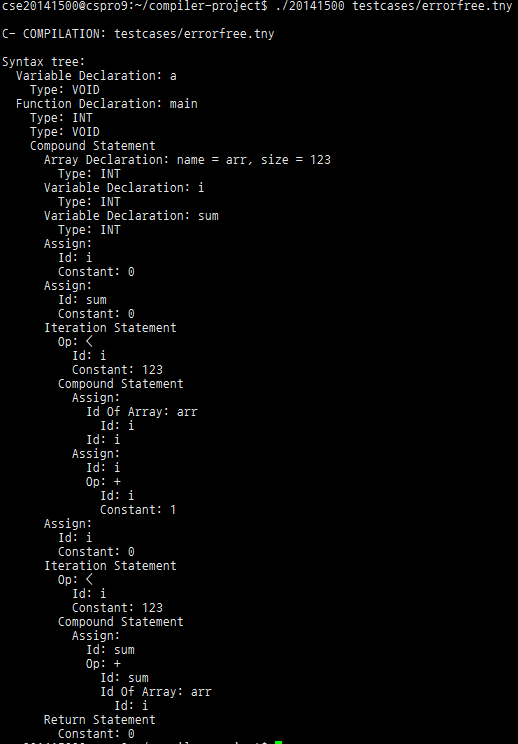


### syntax에러가 있는 소스파일 parsing 시, (error.tny)



위와 같이 몇 line에서 어떤 token에서 에러가 났는 지 띄워준다.

### syntax에러가 없는 정상적인 소스파일 parsing 시, (errorfree.tny)



위와 같이, 생성된 abstract syntax tree의 내용을 가독성있게 출력해준다.

## 라. 시험 내용

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

* C- language 의 syntax를 파싱하기 위한 yacc file을 작성한다. 여기에는 syntax rule, semantic action등이 포함된다.
* 파서의 conflict을 모두 없앤다.
* 결과적으로 프로그램의 abstract syntax tree를 생성하고 이를 가독성있게 출력해준다.

그리고 나는 위 3가지 모두를 달성하였다.

이 것에 대한 증거는 보고서의 '분석 내용'과 '제작 내용'에 적어놓았다.

## 마. 평가 내용

(라)의 시험내용에서 말했듯이 난 3가지 기준을 모두 만족시키는 C- language parser software를 개발하였다. 생성된 파서에 conflict가 존재하지 않고, '분석 내용'에 언급한 여러 가지 문제점들을 해결 했으므로, 내 소프트웨어는 안정적이고 신뢰가 있다고 할 수 있다. 그리고, abstract syntax tree를 가독성 있게 출력했으므로, 사용자 친화적이라 할 수 있다.

# V. 기타

## 가. 자체 평가

앞의 '평가 내용'에 적었던 이유로 인해 나는 좋은 점수를 받아야한다고 생각한다. 그리고 나는 bitbucket을 이용하여 git으로 프로젝트의 형상관리를 수행하였다.

## 나. 느낀점

비록, C언어는 아니지만, C의 subset인 C- language의 parser를 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 만들어 보고 싶다.