**Design and Development of Compiler**

**for C- Language**

**(설계 프로젝트 수행 결과)**

1. **각 단계별 결과 보고서**

**과목명: [CSE4120] 기초 컴파일러 구성**

**담당교수: 서강대학교 컴퓨터공학과 정 성 원**

**개발자: 정재훈 20151607**

**개발기간: 2019. 3. 25 – 2019. 3. 27**

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

Phase 1: Design and Implementation of Lexical Analyzer

제출일 : 2019.03.27

개발자 : 정재훈

I. 개발 목표

C- language로 짜여진 코드를 컴파일하는 컴파일러를 만들기 위하여, 가장 첫번째 단계인 Lexical Analysis를 구현하는 프로그램이다. Lexical Analysis의 주요 기능은 코드 스트링을 받아 코드를 토큰으로 분류해 내는 것이다. 따라서 이번 프로그램은 코드 파일 이름을 입력 받아 lexical analyze하여 라인마다 Line number, Token 그리고 Lexeme을 출력해주는 프로그램을 개발한다.

II. 개발 범위 및 내용

가 개발 범위

이 프로그램을 작성하기 위하여 가장 먼저 각 Token 정보를 담고 있는 header파일인 globals.h를 작성하고 string을 token 단위로 끊어 분석 가능한 DFA 함수를 생성해야 한다. (tiny.l) 이 DFA함수에의해 토큰으로 분류가 되면 토큰에 맞는 Token name과 Lexeme를 출력해주는 함수를 작성해야한다.(util.c) 또한 line number를 출력해야하므로 DFA 함수를 작성시 new line을 고려하여 작성한다. 마지막으로 file name을 받고 이에 대한 출력함수를 부르는 메인함수를 작성해야한다.(main.c)

나 개발 내용

- DFA 함수 작성 및 globals.h 작성

먼저 tokenType을 입력해줘야 하는데 이는 globals.h에 typedef enum으로 미리 정의해준다. 이는 모두 C- language 기준이며 이에 대한 lexical convention은 Compiler construction : Appendix A를 참고하였다.

DFA함수를 작성하기 위해서 flex라는 오픈소스를 이용한다. 여기에서는 정규표현식을 입력하면 flex에서 자동으로 DFA에 해당하는 코드를 작성해준다. 따라서 tiny.l파일에 각 reserved word, special word, number, identifier, comment, newline 등에 관한 정규표현식을 문법에 맞게 작성한다. 또한 lex파일 하단부에는 C파일로된 함수를 작성할 수 있는데 getToken()이라는 함수를 작성하여 DFA함수인 yylex() 함수를 좀 더 이용하기 쉽게 바꾸었다.

- Main 함수 및 Utill 함수 작성

프로그램 인터페이스에 맞게 코드를 토큰단위로 분류하여 이를 라인마다 출력해야하므로 이를 출력하는 함수 printToken()을 util.c에 작성한다. 또한 Main함수에서는 End of File이 나올 때까지 모든 코드 string을 getToken()으로 분석한다.

III. 추진 일정 및 개발 방법

가 추진 일정 .

2019.03.25 ~ 2019.03.25 Lexical Analysis 코드 작성

2019.03.26 ~ 2019.03.27 보고서 작성

나 개발 방법

- DFA 함수 작성 및 globals.h 작성

Compiler construction : Appendix A.1 를 참고하여, 토큰들을 Keywords, Special Symbols, ID, NUM,White space, Comment으로 나누고 각 표현에 해당하는 정규표현식을 작성한다. 이제 lineno은 전역 정적 변수로 두어 개별적 new line을 인식하거나 또는 comment에서 new line을 인식하는 경우 늘리는 형식으로 기록한다. 이때 인식될때마다 tiny.l에서 토큰을 반환해줘야하는데 이는 globals.h에 typedef enum으로 모든 토큰들을 입력해주고 이를 반환하도록 해줘야한다. 연속적인 문자열이 토큰으로 들어오는 경우가 있는 반면에 comment는 /\* ..... \*/으로 DFA로 다루기 까다로운 경우이다. 이때는 /\*만 인식하고 뒤에는 C코드를 이용하여 판별하는 식으로 바꾼다. getToken()함수는 DFA함수 yylex()를 이용한 함수로 printToken()까지 호출해주는 함수이다. flex의 내부 변수 yytext등을 이용하여 토큰 인식 및 출력이 한번에 이뤄질 수 있도록 작성한다.

- Main 함수 및 Utill 함수 작성

main함수는 Filename을 Arguement로 받아 이를 열어주고 End of File이 나올때까지 getToken()을 불러 코드의 모든 token을 분석할수 있게 작성한다. Utill.c에서 필요한 것은 printToken()함수로 이는 토큰의 TokenName 그리고 그에 해당한 Lexeme를 출력해주는 함수이다. 이때 각 Reserved Word, Special Word, ID, NUM 등에 따라 출력되는 것이 다르므로 Switch문을 이용하여 분류하여 출력할 수 있도록 한다.

IV. 연구 결과

1. 합성 내용:

tiny.l

lex.yy.c

yylex()

C- File

test1.c test2.c

globals.h

utill.c

main.c

FLEX

C compiler

getToken()

OUTPUT

(Tokenized String)

make 명령어를 입력하면 오픈소스 flex를 이용하여 tiny.l이 lex.yy.c로 바뀌며 그 후 main.c util.c와 함께 gcc를 통해 컴파일 되어 20151607 이라는 실행파일이 생기게 된다. 이를 실행시키기 위해서는 ./20151607 test1.c를 입력하면 되는데, 이를 실행시키면 test1.c에 적힌 모든 string을 lexical analyze하여 tokenized string을 형식에 맞추어 standard output으로 출력하게 된다.

2. 분석 내용:

- globals.h, tiny.l

globals.h에 typedef enum으로 각 토큰을 enumeration 시켜준다. 이때 comment error 역시 토큰 COMERR로 저장해야한다.

Keyword와 Special Symbol은 단일 스트링 “~”으로 표현하고, ID = {letter}+, NUM = {digit}+, white space는 \t과 공백을 포함하며 comment는 c언어로 분류하도록 한다. 이때, FLEX의 규칙에 의해 ID가 keyword보다 아래에 작성되어야 우선순위를 정할 수 있다. comment는 multiple line으로 작성되므로 comment 안에 개행문자가 있을 수 있는데 이를 처리하는 코드를 작성하여 lineno 변수를 유지하도록 한다. 이때 comment error가 날 경우 COMERR라는 토큰을 반환하도록 한다. yylex()함수를 좀 더 편하게 다루기 위하여 getToken()함수를 작성한다. getToken()함수는 yylex()함수를 호출하며, 식별된 Token에 따라 printToken()을 호출하여 Token Name과 Lexeme을 출력할 수 있도록 한다.

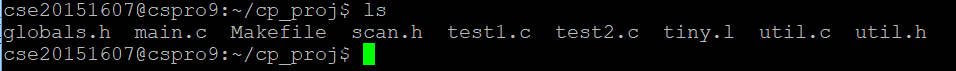
- main.c, util.c

main함수는 argument로 파일 이름을 받도록 하고 이를 통해 해당 파일을 open한다. 해당 코드 파일을 getToken()함수를 통하여 End of FIle이 나올때까지 호출한다.

util.c는 printToken() 함수가 정의되어 있는데 이는 token type과 token string을 받아 해당하는 Token name과 Lexeme을 출력해주는 함수이다. 이 경우 Lexeme는 token string이 되며 Token name은 전역으로 따로 선언 해주어야한다. 따라서 const char \*tokenID[] 라는 변수를 선언하고 여기에는 Token에 해당하는 Token name을 각 enumeration에 맞게 저장한다. 그렇게 하면 각 Token Type에 맞는 enum 번호에 따라 tokenID[number]을 통하여 쉽게 tokenName을 출력할 수 있다. 이때 reserved word와 special symbol은 출력형식이 조금 다른데 reserve word는 Token Name과 Lexeme를 출력하고 special word는 둘다 lexeme를 출력해주면 된다. END OF FILE은 그냥 EOF를 출력하고 comment Error는 ERROR comment Error를 출력한다.

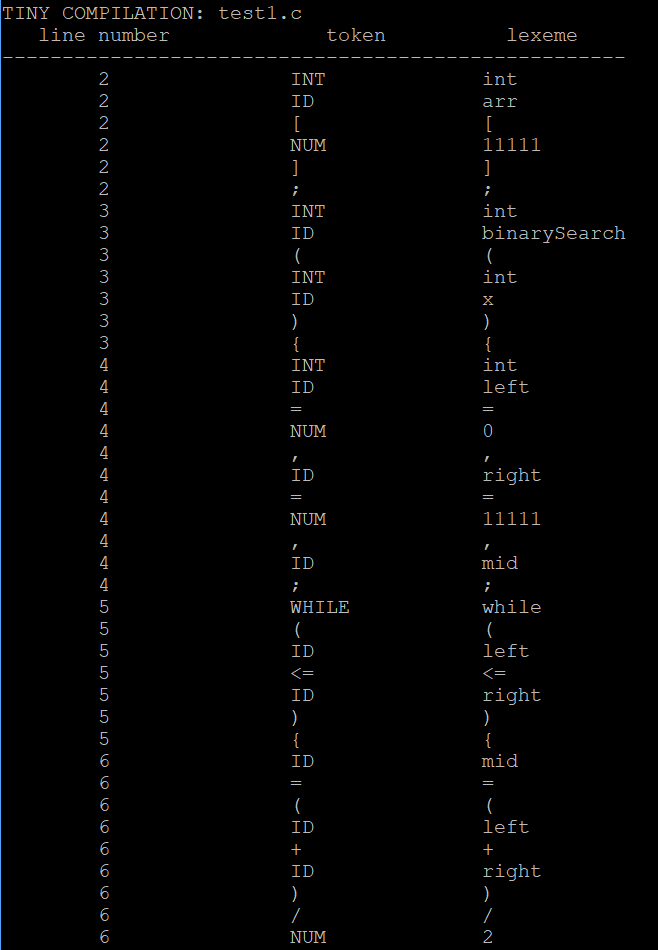
3. 제작 내용:

파일 이름이 입력되면 해당 파일에 대한 Lexical Analysis를 수행하는 프로그램을 제작하였다. 코드는 main.c util.c globals.h util.h tiny.l scan.h를 작성하였다.

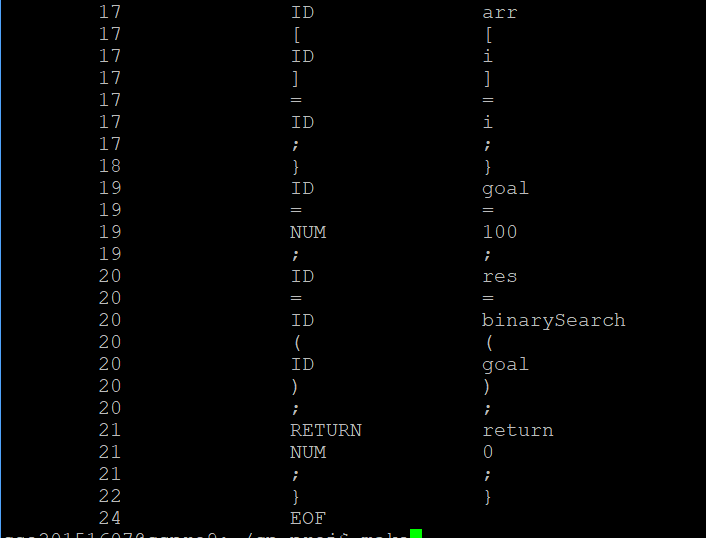


4. 시험 내용

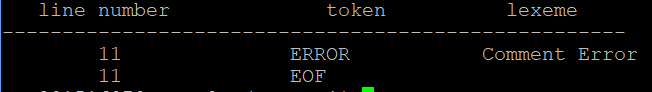
(test1.c)



=중간생략=



(test2)



test1,2 모두 올바르게 출력되었다.

5. 평가 내용:

내가 개발한 소프트웨어는 Lexical Analysis로 C- Language의 Lexcial convention에 따라 모든 토큰을 1 Pass만에 Identify한다는 장점이 있다. 또한 Flex 오픈 소스를 활용하여 최적 state의 DFA를 구현했으며 이를 통해 속도와 공간적인 장점을 극대화시켰다. 단점이라고 한다면 Flex가 수행하는 모든 과정을 예측하는데 어려움이 있기에 이에 대한 에러처리가 취약하다.

V. 기타

1. 자체 평가:

원래 교재속 comment 처리는 line number에 대한 처리가 들어가 있지 않으나 Flex 문법 작성시 comment 토큰 처리할때 line number 처리 코드를 추가하여 정확한 Line number 처리를 구현하였다. 또한 구조적인 코드 작성과 Makefile 작성으로 모듈화와 관리 유지가 쉽도록 하였다.

2. 느낀점

본 프로젝트를 진행하면서 Compiler 5단계중 첫단계인 Lexical Analysis의 원리와 구현에 대해 더 정확하게 알 수 있었고, 반드시 Regular Expression으로 인식하기 보다는 C code를 다소 활용하여 구현하게 된다면 좀 더 쉬운 토큰 인식이 가능함을 알게 되었다.