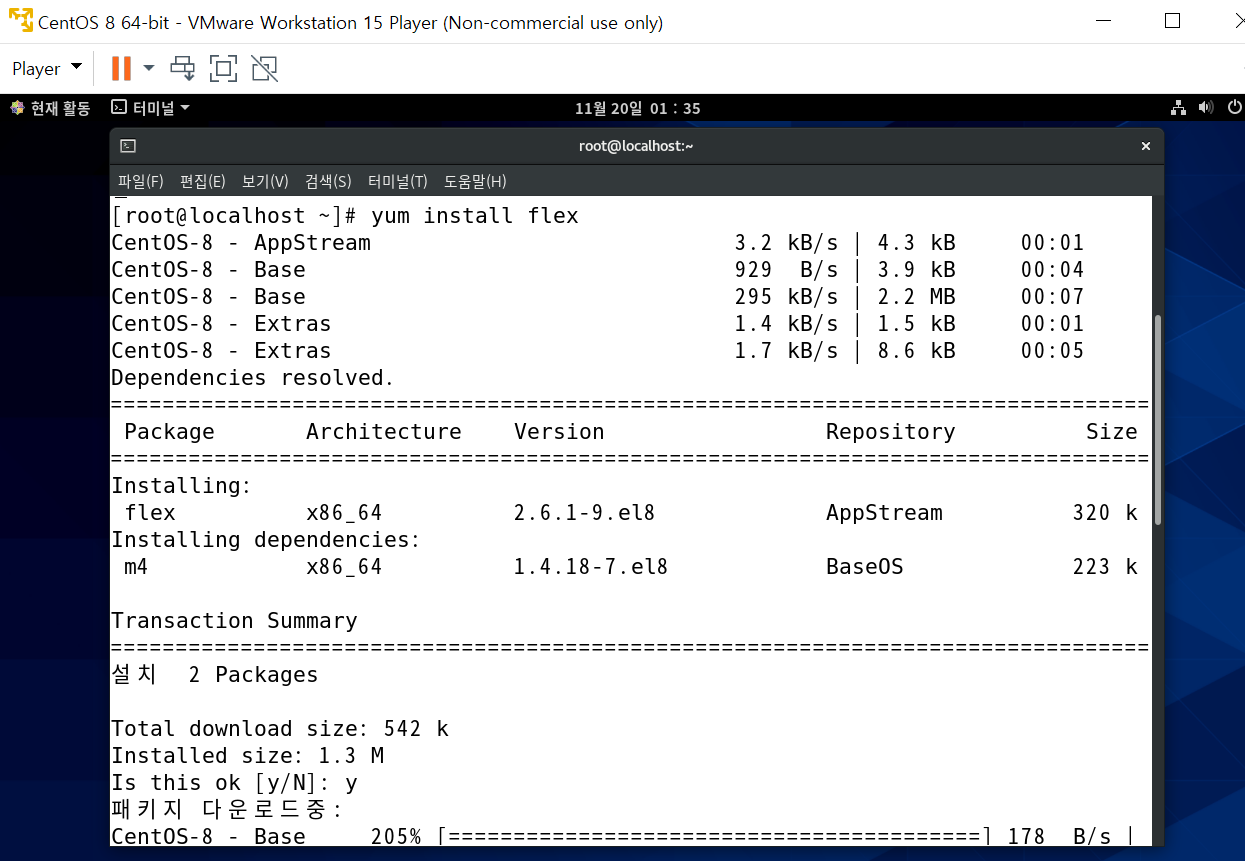
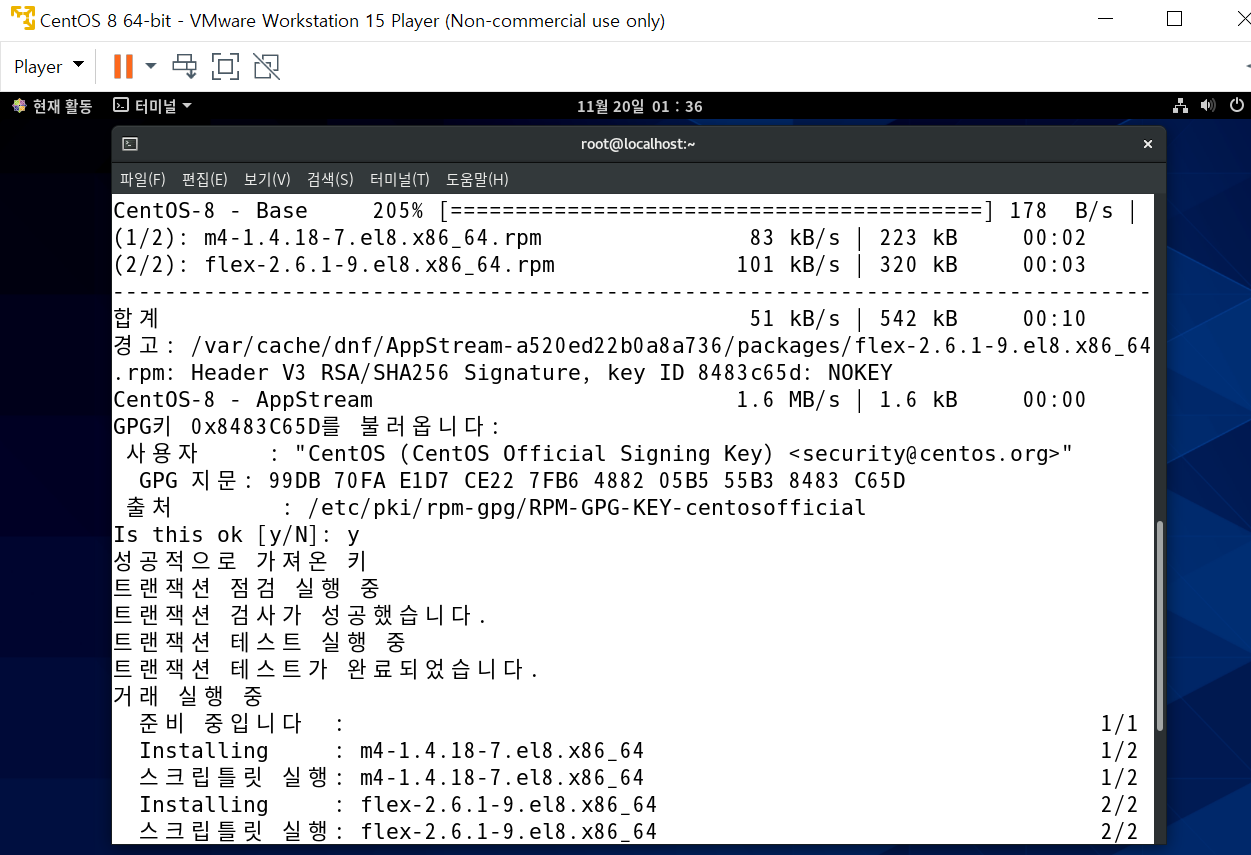
개발 환경: Cent OS Linux

1팀 : 권지현(팀장), 김동진, 윤성일, 이다인, 이현석

작성 코드 언어: C언어 (교재 160p-163p 참고)

**\* Lex 설치**







**-- lex 설치 완료 --**

**(1) Lex의 입력파일: test.l**

%{

#include <stdio.h>

#include <stdlib.h>

enum tnumber { TEOF, TIDEN, TNUM, TASSIGN, TADD, TSEMI, TDOT, TBEGIN, TEND, TERROR};

%}

letter [a-zA-Z]

digit [0-9]

%%

begin return(TBEGIN);

end return(TEND);

{letter}({letter}|{digit})\* return(TIDEN);

":=" return(TASSIGN);

"+" return(TADD);

{digit}+ return(TNUM);

";" return(TSEMI);

\. return(TDOT);

[ \t\n] ;

. return(TERROR);

%%

void main()

{

enum tnumber tn;

printf("Start id Lex\n");

while((tn=yylex()) != TEOF){

switch(tn){

case TBEGIN : printf("Begin\n"); break;

case TEND : printf("End\n"); break;

case TIDEN : printf("Identifier: %s\n", yytext); break;

case TASSIGN : printf("Assignment\_op\n"); break;

case TADD : printf("Add\_op\n"); break;

case TNUM : printf("Number: %d\n", atoi(yytext)); break;

case TSEMI : printf("Semicolon\n"); break;

case TDOT :printf("Dot\n"); break;

case TERROR : printf("Error: %c\n", yytext[0]); break;

}

}

}

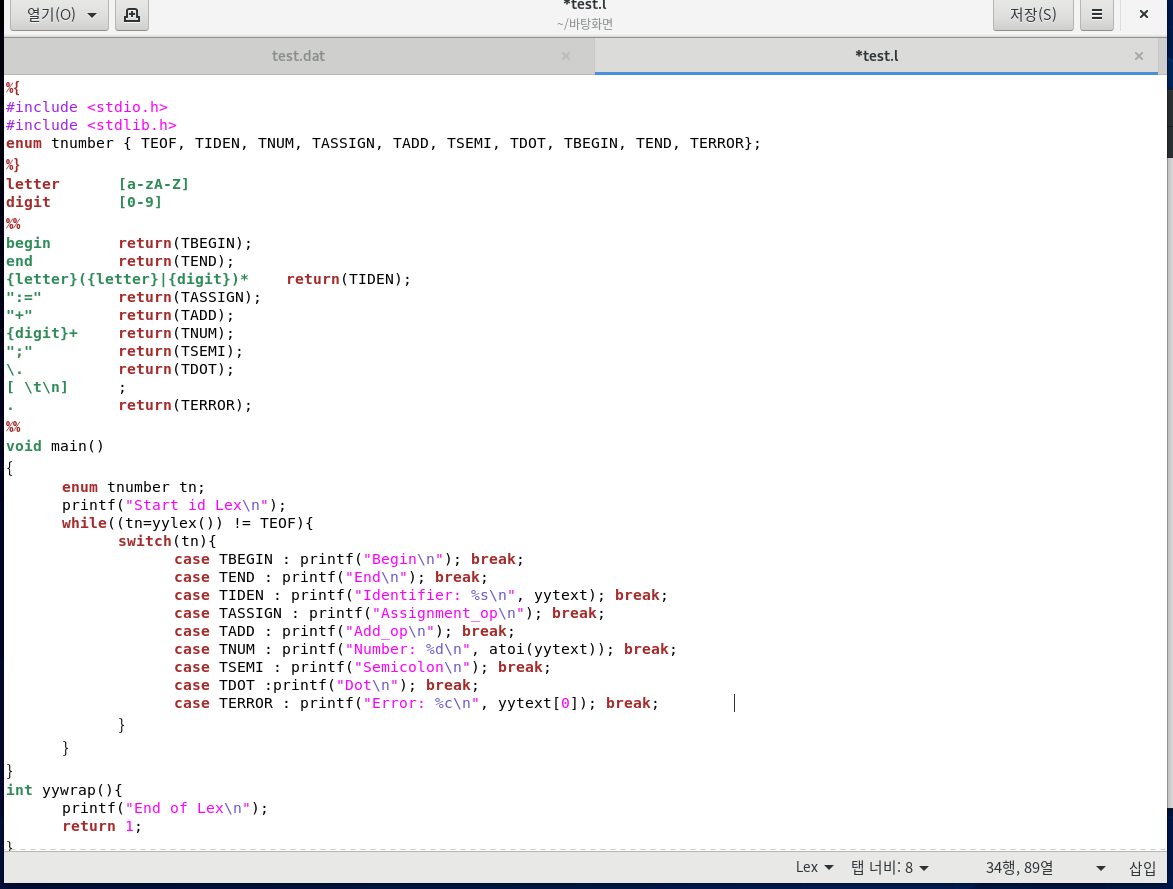
int yywrap(){

printf("End of Lex\n");

return 1;

}

**\* 실제 작성 파일**



**(2) 입력스트림 파일(데이터 파일): test.dat**

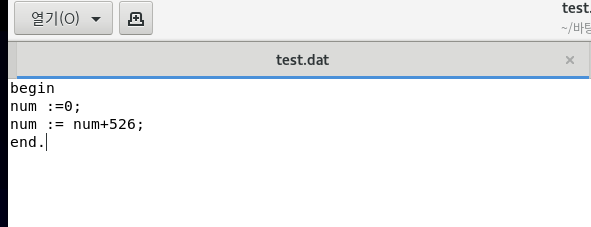
begin

num :=0;

num := num+526;

end.

**\* 실제 작성 파일**

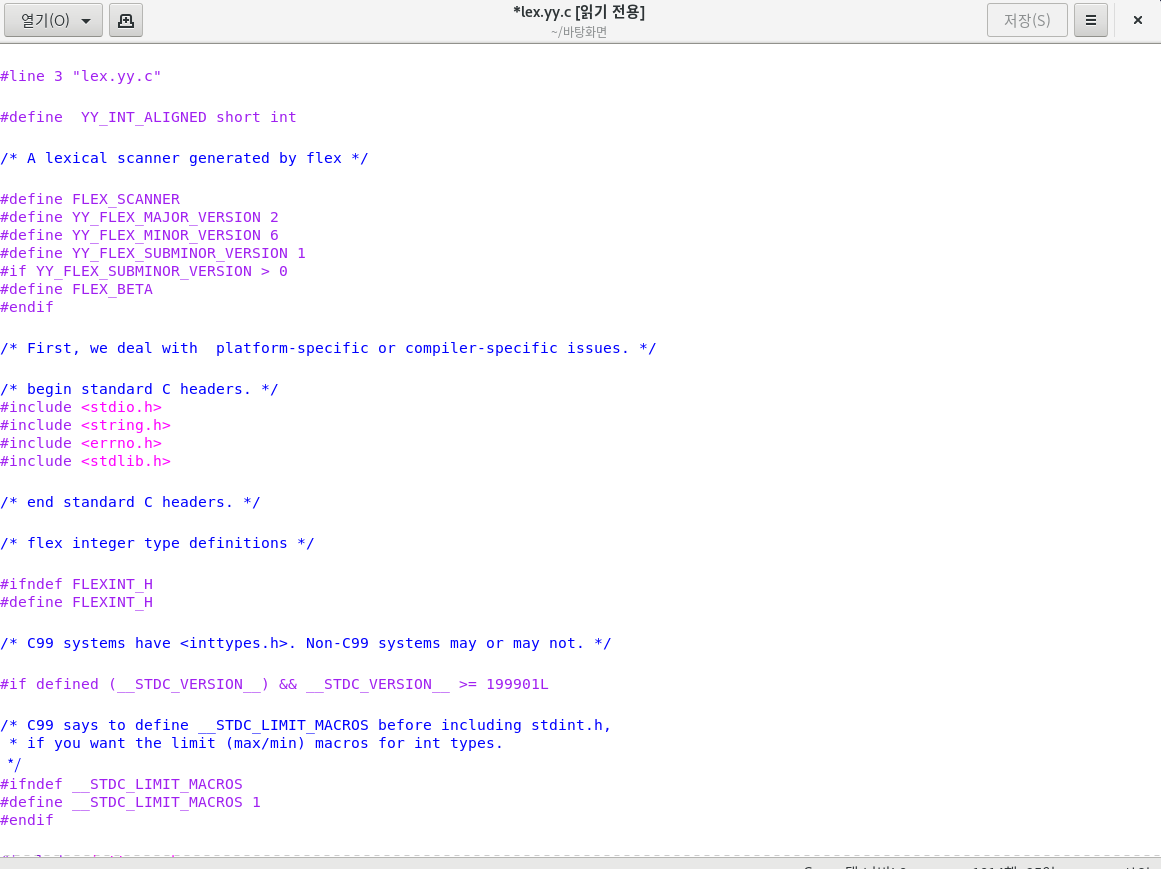


**(3) LEX가 생성한 스캐너 파일: lex.yy.c**

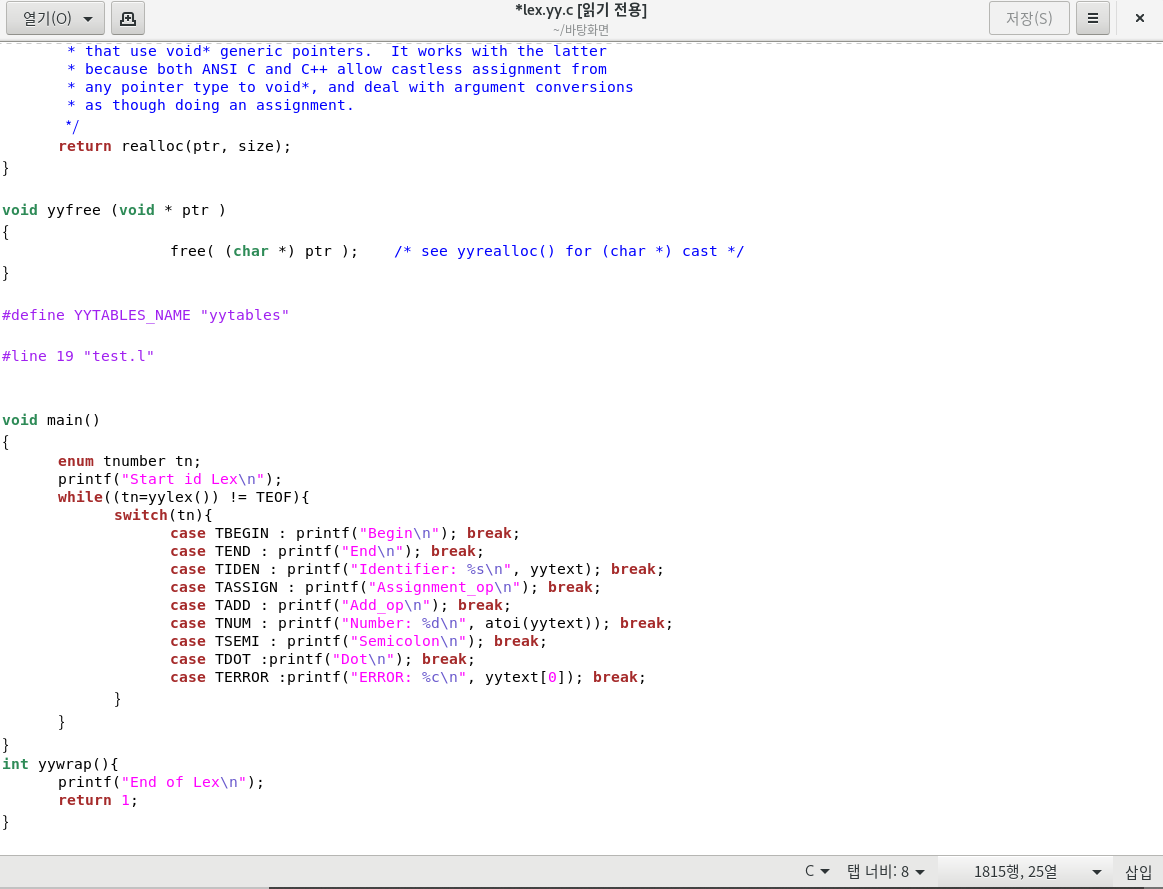
코드의 양이 70p(약 1825행)가 넘어가기에 .c파일로 첨부하였습니다.

해당 화면은 lex.yy.c의 첫번째 페이지와 마지막 페이지입니다.

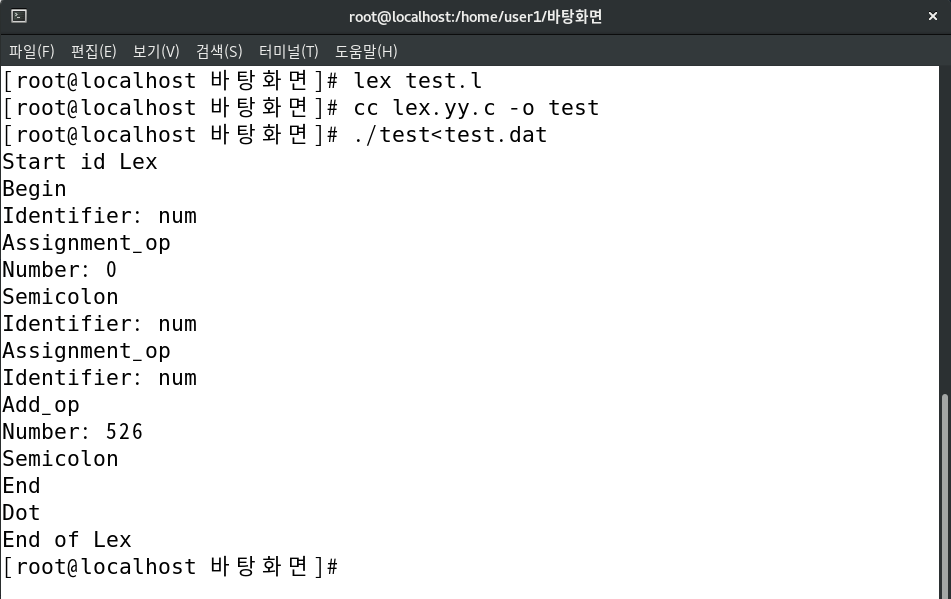
**\* 첫번쨰 페이지**



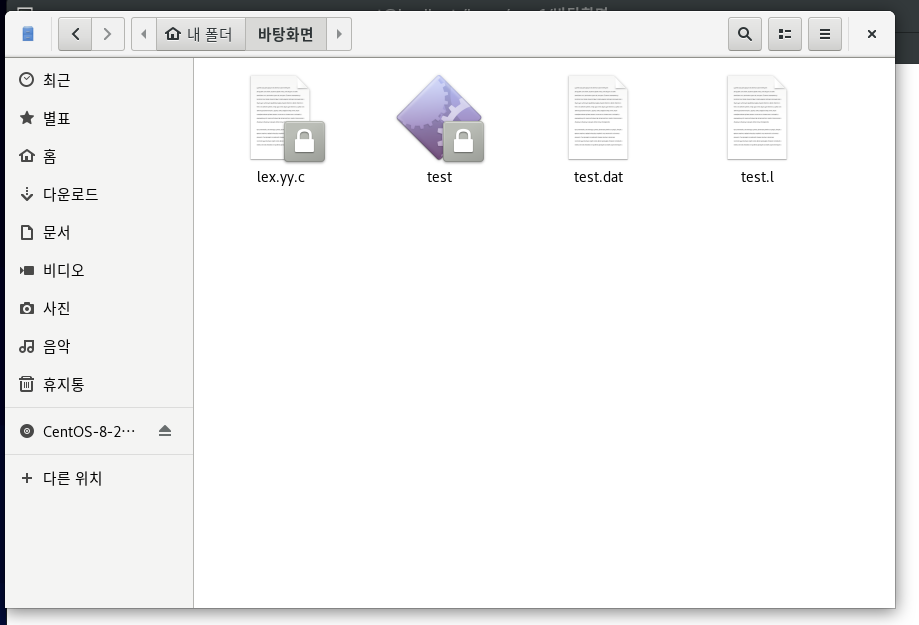
**\* 마지막 페이지**

****

**\* 실행결과 화면**



**\* 코드 및 파일 작성 디렉토리: Cent OS의 바탕화면**



**소감문**

[ 권지현 ]

사전에 스캐너의 생성 및 동작원리를 이해하고 공부한 후 어휘분석기를 직접 구현해보는 과정에서 어휘분석기에 대한 이해도가 더 높아졌습니다. C언어로 작성한다는 점이 이전에 구현했던 어휘분석기와 비슷한 점이 있었습니다.

하지만 이번에는 리눅스를 이용하였기에 구현 방식과 코드 구성에 있어 차이점을 느꼈습니다. 명령어를 통해 리눅스에 직접 lex 패키지를 설치한 후 확장자 명이 \*.l 인 Lex 파일과 데이터 파일 즉, 입력 파일인 test.dat를 작성하고 저장하였습니다. 이후 161.p를 참고해 명령어를 치니 자동으로 lex.yy.c 가 생성되는 것을 확인할 수 있었으며, 입력 파일에 대한 실행결과 또한 정상적으로 출력되는 것을 확인하였습니다. 처음부터 끝까지 직접 작성해야 했던 이전 과제와 달리 리눅스에서 명령어를 통해 Lex를 설치하고 규칙을 정하면 자동으로 lex.yy.c가 생성되는 걸 보아 오류가 날 확률이 더 적었던 것 같습니다.

큰 오류메시지 없이 정상적으로 잘 동작하게 구현하였다는 점에서 매우 뿌듯하였습니다. 또한 다양한 방법으로 어휘분석기를 구현하는 과정을 거치니 스캐너 동작원리의 이해에 있어 한 층 더 심도 있게 알게 되었습니다.

[ 윤성일 ]

평소 lex 어휘분석기에 관심은 있었지만 어렵고 복잡해서 다가가기 어려웠는데 팀프로젝트를 통해서 쉽게 접근할 수 있었고 만족스러운 팀 프로젝트였습니다.

[ 이다인 ]

실습에는 자신이 없어 힘이 들었지만 팀장님이 많이 도와주시고 팀을 잘 이끌어 주셔서 끝까지 해낼 수 있었습니다. 이번 프로젝트를 통해 lex 어휘분석기에 대한 관심도 많이 상승하여 앞으로의 수업에 더욱 흥미롭게 참여할 수 있게 된 것 같아 좋은 경험이었습니다.

[ 김동진 ]

lex 어휘 분석기의 활용도에 대해서 이론으로만 배웠을 때 어떤 식으로 구현이 되는지 사실 조금은 이해가 안가는 부분도 있었는데 실습을 통해서 구현되는 코드를 보니 이해가 되었고, 어휘 분석기라는 것의 역할과 활용도를 알고 나니 흥미가 생겨서 앞으로 수업을 들을 때 더 잘 이해하고 잘 활용할 수 있을 것 같다고 느꼈습니다.

[이현석]

과제1과 과제2를 하며 어휘 분석기를 생성하는 방법에 범용 프로그래밍 언어를 사용하여 작성하거나 어휘 분석기 생성기로부터 얻을 수 있는 두 가지의 방법이라는 것을 알게 되었다.

이번 과제는 어휘 분석기 생성기 중에 가장 많이 사용되고 유명한 LEX를 이용하여 스캐너를 작성하는 과제였다. 첫 번째 과제 때는 c언어를 이용하여 프로그래밍을 하여서 작성하였기 때문에 시간이 조금 걸렸지만 이번 LEX를 통해 작성된 결과물은 이미 라이브러리가 존재했기 때문에 자동으로 C코드를 생성하고 생성된 C파일로 어휘 분석을 하는 것이 더 편리하다는 것을 알게 되었고, 이는 간편하고 짧은 시간 내에 토큰들을 추출할 수 있었다.

그리고 LEX를 활용하기 위해 윈도우가 아닌 리눅스 환경해서 실행했고 리눅스 환경에서 LEX를 이용하려면 LEX 라이브러리를 이용하여야 한다는 사실을 알게 되었다. 그래서 리눅스 명령어를 이용하여 LEX 라이브러리를 설치하고 LEX 라이브러리에 대한 공부를 할 수 있었다.

이렇게 두 가지 방법을 통해서 직접 실습하고 만들어보고 조별 과제를 한 결과 어휘 분석 단계에 조금 더 심층적으로 알게 되었고, 이는 컴파일러를 공부하는데 많은 도움이 되었다. 또한 이번 과제를 통해서 컴파일러의 전단부 과정인 어휘 분석과정에 대해 확실하게 더 알게 되었고, 리눅스에 대한 공부를 할 수 있어 많은 도움이 되었고 실행하는 과정에서 많은 재미를 느낄 수 있었다. 즉, LEX 과제를 하면서 다시 한 번 컴파일러의 어휘 분석에 대해 파악할 수 있었다.

컴파일러가 제대로 공부되지 않은 상태서 시작했던 이번 과제는 친구 들과의 협업과 이번 팀 프로젝트를 통하여 조금은 심도 있게 접근하고 공부할 수 있어서 뜻 깊었다. 내가 부족하면 그만큼 주변인에게 도움이 되지 못하기 때문에 도움이 되기 위해 자발적으로 준비를 하는 내 모습을 발견할 수 있었고, 공부를 하면 그 만큼 알게 된다는 것 또한 느낄 수 있었다.

앞으로 남은 기말고사까지의 범위를 조금 더 예전보다 흥미 있게 공부하고 준비할 수 있을 것이라는 자신감 또한 얻게 되었다. 앞으로 더욱 심도 있게 공부하고 준비하여 컴파일러뿐 만 아닌 모든 과목에서도 이러한 자세로 호기심을 갖고 결과를 얻어낼 것이다. 흥미와 자신감을 갖게 해준 이번 과제에 고마움 또한 느끼게 되었다. 또한 이러한 과제를 내주신 교수님께 감사하다.