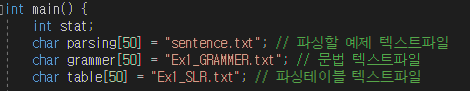
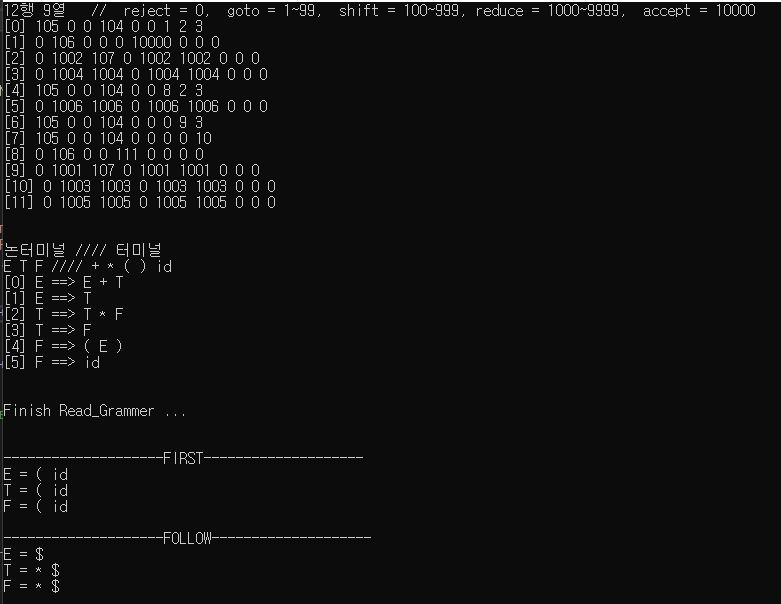
**Compiler: Assignment 1 결과보고서**

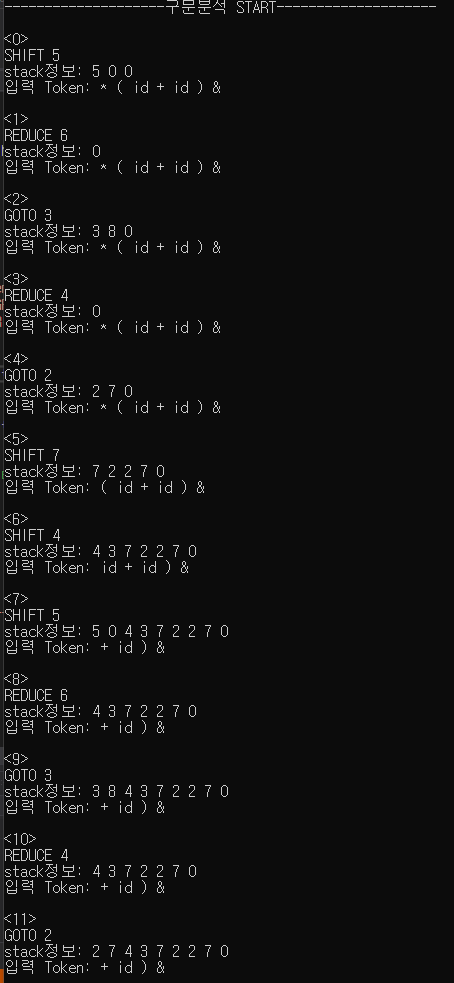
**컴퓨터공학부 201614868 이성민**

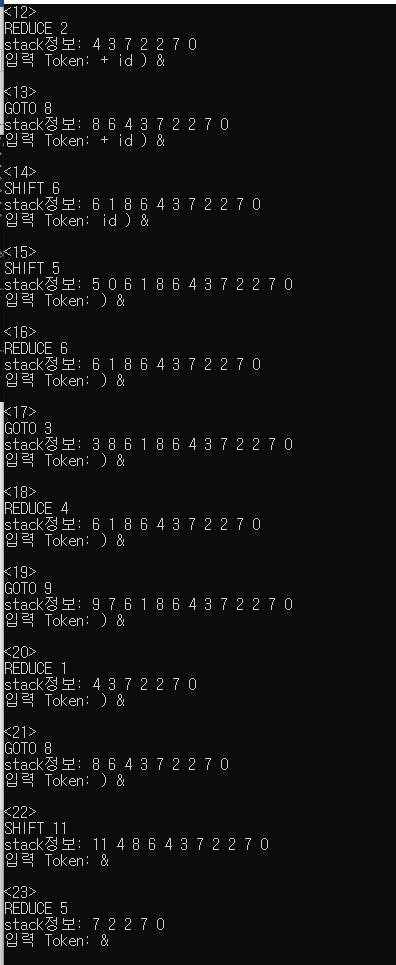
**EX1)**

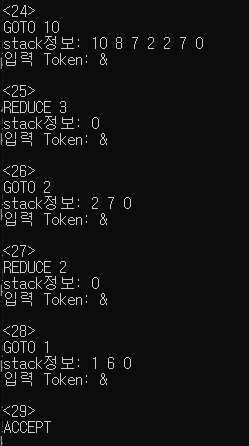
1. **SLR구문분석**

****

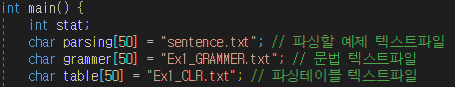


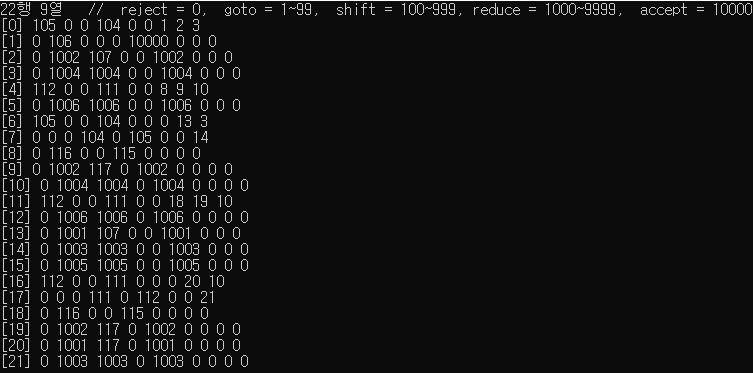


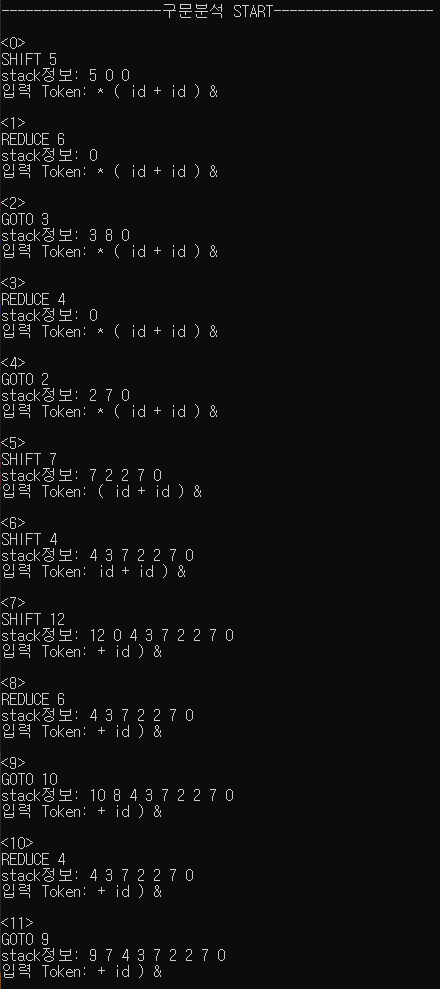


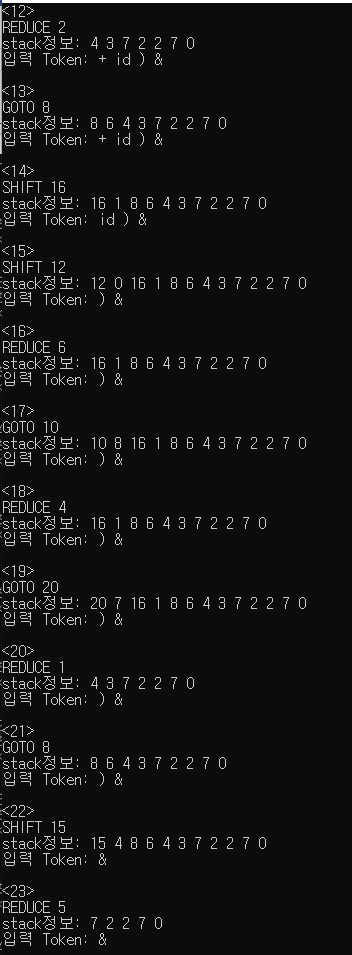


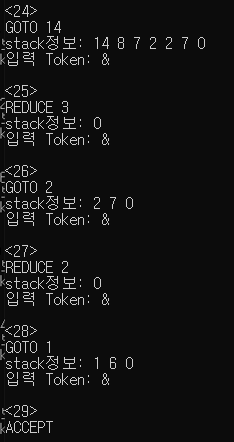
1. **CLR구문분석**



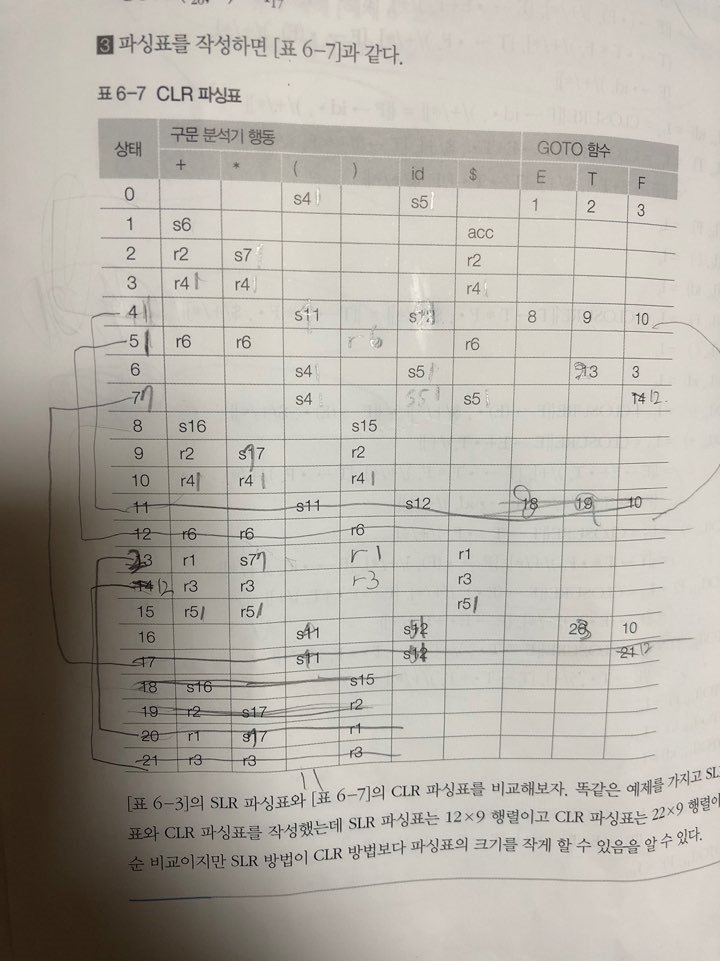








1. LALR구문분석



CLR파싱표에서 lookahead만 다르고 같은 state들을 합쳐서 파싱테이블을 간소화시킨다.

151 0 0 141 0 0 1 2 3

0 106 0 0 0 10000 0 0 0

0 1002 177 0 0 1002 0 0 0

0 1041 1041 0 0 1041 0 0 0

151 0 0 141 0 0 8 9 10

0 1006 1006 0 1006 1006 0 0 0

151 0 0 141 0 0 0 23 3

151 0 0 141 0 151 0 0 12

0 116 0 0 115 0 0 0 0

0 1002 177 1002 0 0 0 0

0 1041 1041 0 1041 0 0 0 0

0 1001 177 0 1001 1001 0 0 0

0 1003 1003 0 1003 1003 0 0 0

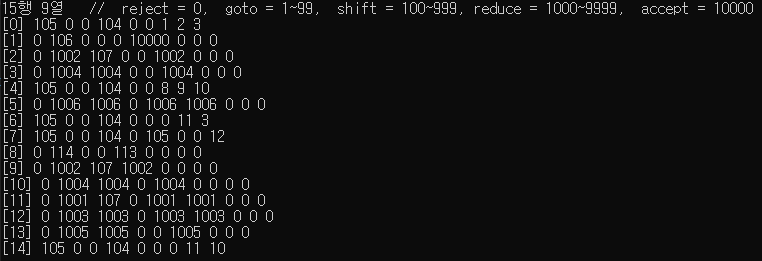
0 1051 1051 0 0 1051 0 0 0

151 0 0 141 0 0 0 23 10

다음과 같이 되는데,

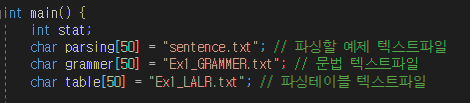
c코딩을 각 줄을 0,1,2,3,4,5…. 이렇게 연속적인 상태로 입력을 받게 되어서 41-> 4, 51->5 이런 형태로 순차적인 상태표를 그릴 수 있게끔 다시 변환한다.

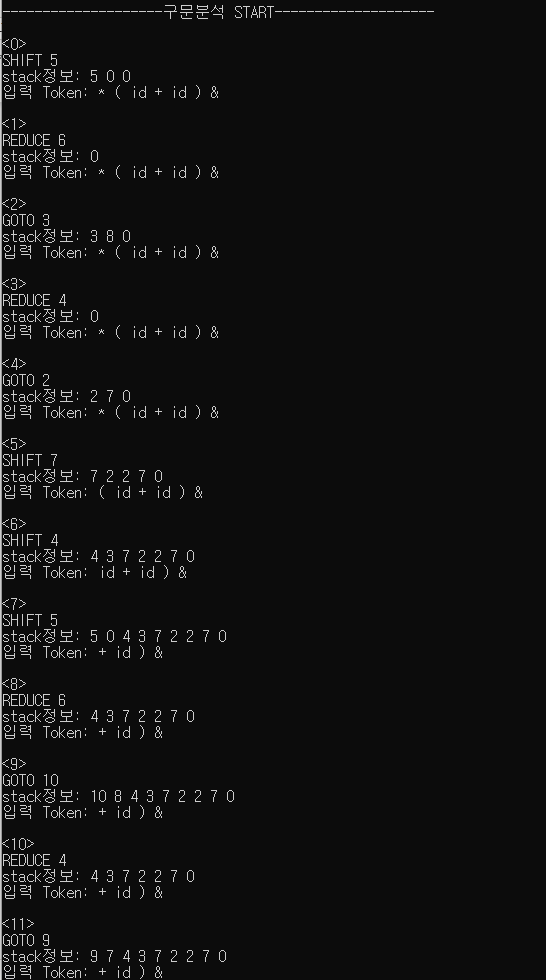
그렇게 하면,

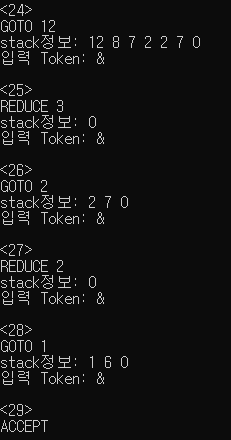
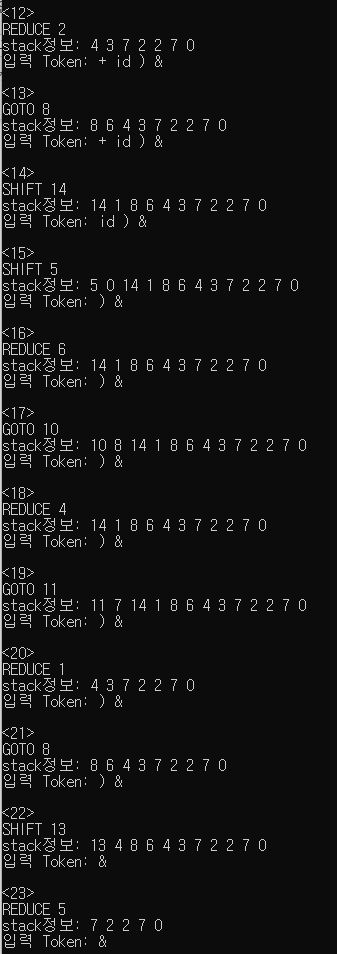


다음과 같은 테이블을 얻을 수 있다.

다음 캡쳐 화면은 구문분석 결과이다.

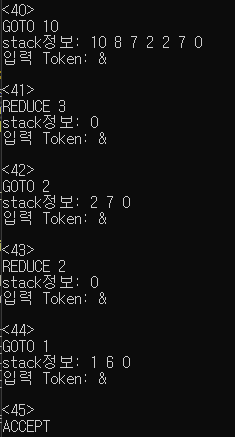




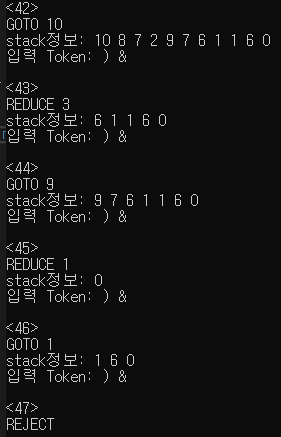


지금까지는 id \* ( id + id )를 테스트했고

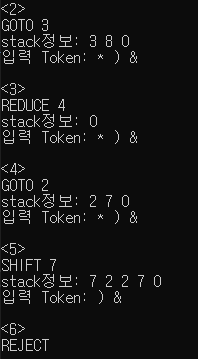
id \* ( id + ( id + id ) ) 테스트시에



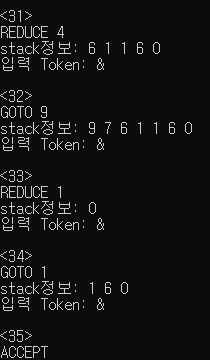
id \* id + id \* id + id \* id \* id ) 테스트시



id \* ) id 테스트시



id \* id + ( id \* id ) 테스트시

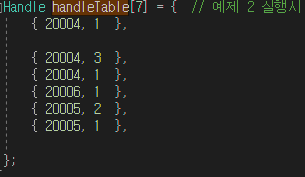


다음과 같이 결과가 나온다.

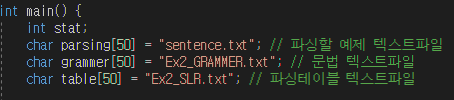
**EX2)**

**예제 2 코드 구문분석시 추가해줘야할 코드(시간이 없어서 자동입력 추가못함)**

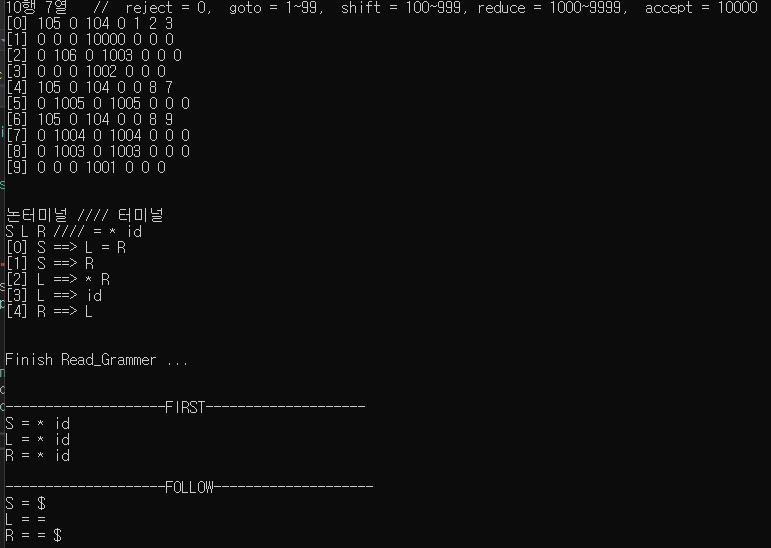
**//그리고 입력할 문장 텍스트 파일에서 받아올때 맨 뒤 문자를 못가져 오는 경우가 생길 수 있는데 뭐때문인지 모르겠지만 \* id = id 를 입력하고싶으면 \* id = id id 이렇게 끝에 하나 더 붙여줘야한다 .. 에러처리를 못했다.**

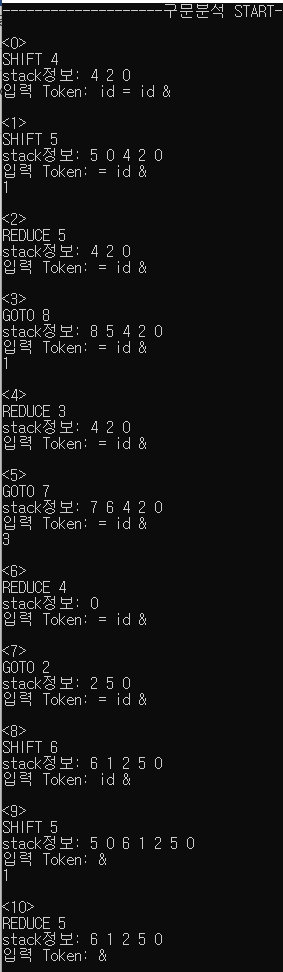
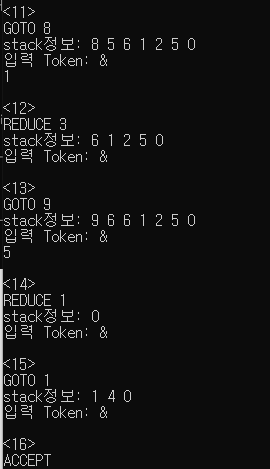
****

****

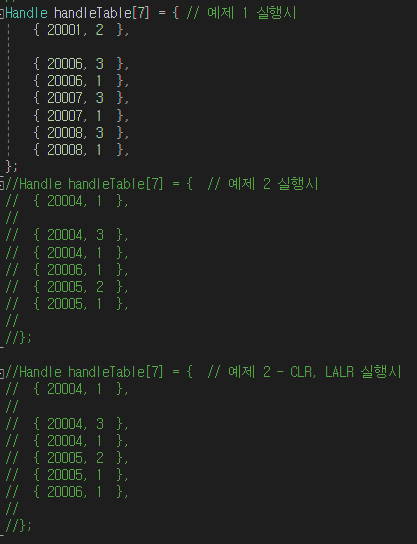
****

1. **SLR구문분석**

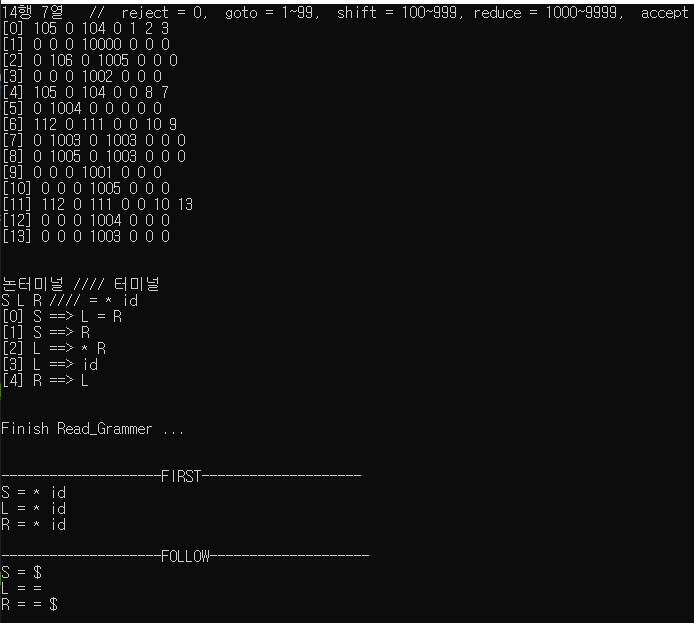


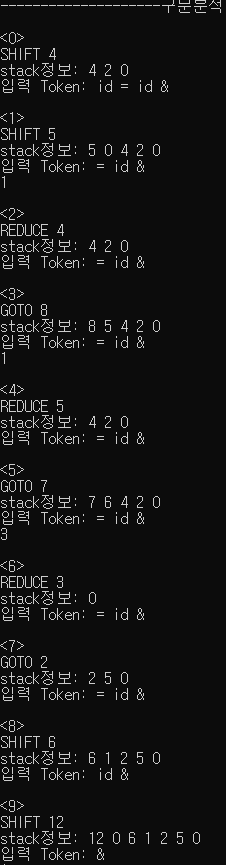
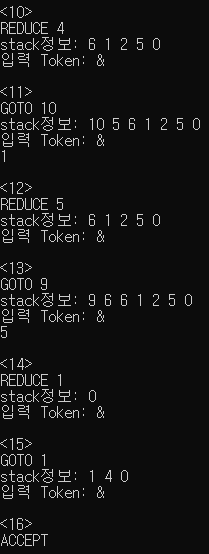
 

1. **CLR구문분석**

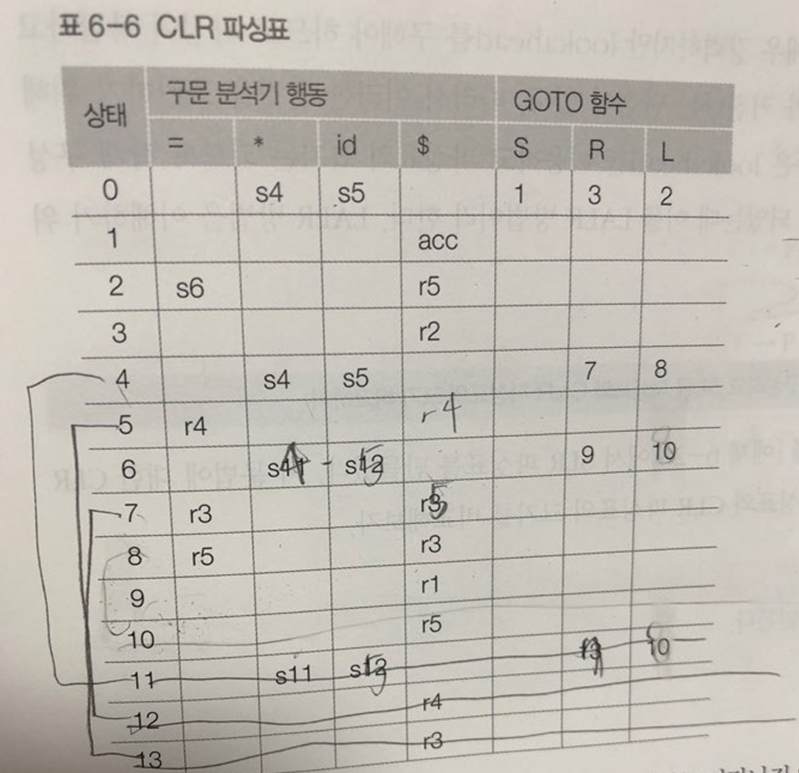
****

**CLR, LALR테이블로 구문분석하려면 또 변경해 주어야 할 사항들이 있다.( 테이블을 일정하게 짜지 않아서,, )**

****

**** ****

1. **LALR구문분석**

****

CLR파싱표에서 lookahead만 다르고 같은 state들을 합쳐서 파싱테이블을 간소화시킨다.

**105 0 104 0 1 2 3**

**0 0 0 10000 0 0 0**

**0 106 0 1005 0 0 0**

**0 0 0 1002 0 0 0**

**105 0 104 0 0 8 7**

**0 1004 0 1004 0 0 0**

**105 0 111 0 0 10 9**

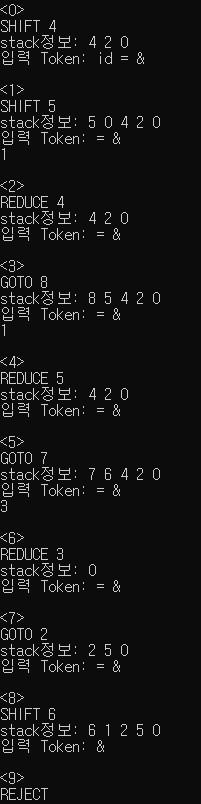
**0 1003 0 1003 0 0 0**

**0 1005 0 1003 0 0 0**

**0 0 0 1001 0 0 0**

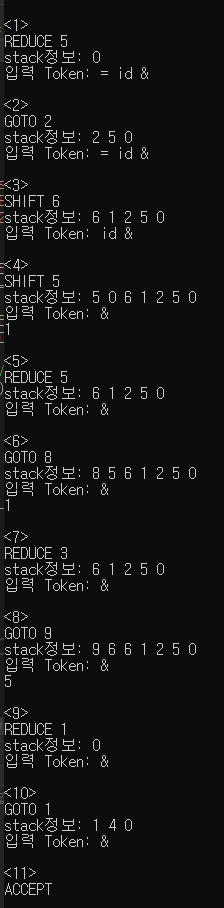
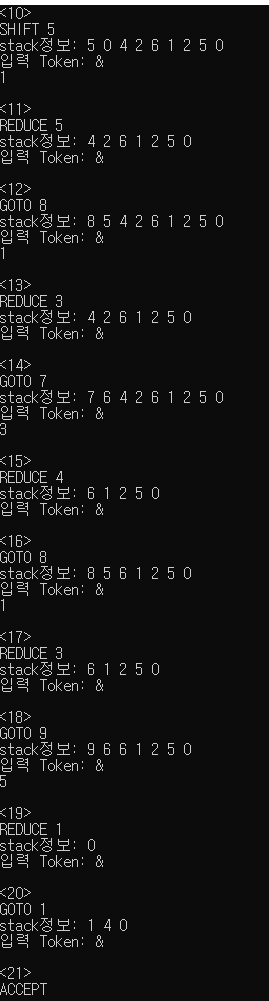
**0 0 0 1005 0 0 0**

**구문분석을 해보면,**

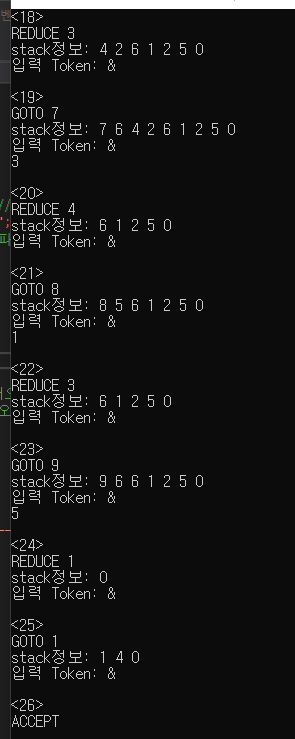
****

**잘되는걸 볼 수 있다.**

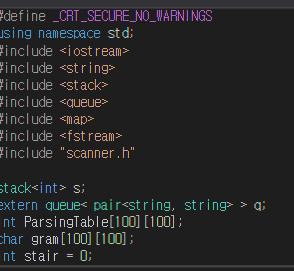
**문장) Id = id id \* id = \* id 구문분석**

**** **** 

= id ) \* \* id = \* id 구문분석

**** ****

**#그 외 설명**



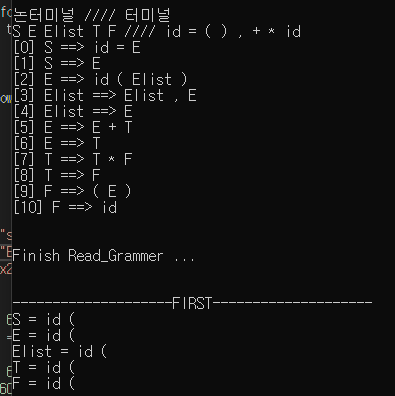
C++ STL의 스택을 통해 구문분석 자료구조로 사용하였고, 큐를 통해 구문분석할 식을 string 토큰으로 저장해서 쉽게 빼서 사용할 수 있게 만들었다.

소스코드는 **소스.cpp**와 **Scanner.h**

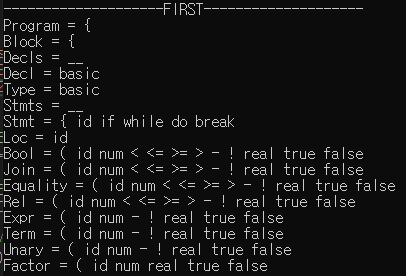
로 구성되었는데, 소스.cpp에는 메인 코드와, first, follow, 파일입출력, 구문분석코드 다 포함되어있고,

Scanner.h 에는 구문분석할 문장을 스트링 토큰 단위로 나누어 큐에 저장할 수 있게하는 함수를 넣었다.

First, Follow는 재귀를 이용해서 구현했는데, First같은 경우는



<<예제3



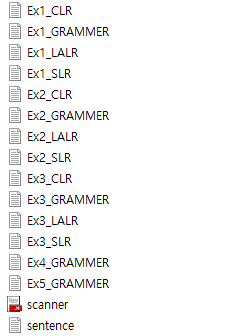
<<예제4 까지 잘 출력하지만,

Follow 간단한 예제는 출력하는데 조금만 꼬우면 작동을 잘 안한다. 수정이 필요하다.

 << follow 계산함

예제3, 4 파싱테이블은 만들지 못했고,

그리고 파스트리 출력도 구현하지 못하였다.



각 예제별 문법, 파싱테이블, 입력할 문장(sentence) 텍스트파일이다.

이 텍스트파일을 입력받아 코드를 수행한다. 코드 파일에 포함되어있다.