http://pl.skuniv.ac.kr

컴파일러 입문

제 8 장

LR 구문 분석

목 차

8.1 LR Parser

- an efficient ***Bottom-up*** parser for a large and useful class of context-free grammars.
- the "L" stands for ***left***-to-right scan of the input;
  the "R" for constructing a ***Rightmost*** derivation in reverse.

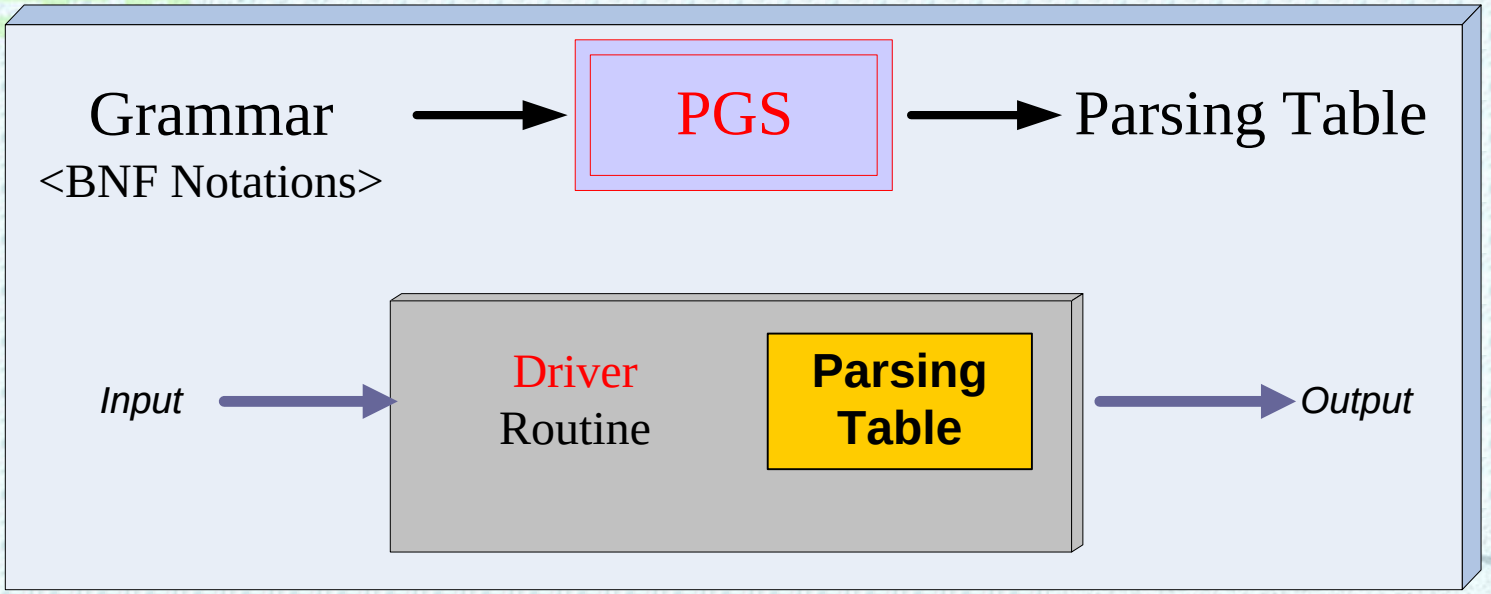
- The attractive reasons of LR parsers
  (1) LR parsers can be constructed for ***most*** programming languages.
  (2) LR parsing method is more ***general*** than LL parsing method.
  (3) LR parsers can detect ***syntactic errors*** as soon as possible.

But,

- it is too much work to implement an LR parser ***by hand*** for a typical programming-language grammar.

=====> ∴ Parser ***Generator***

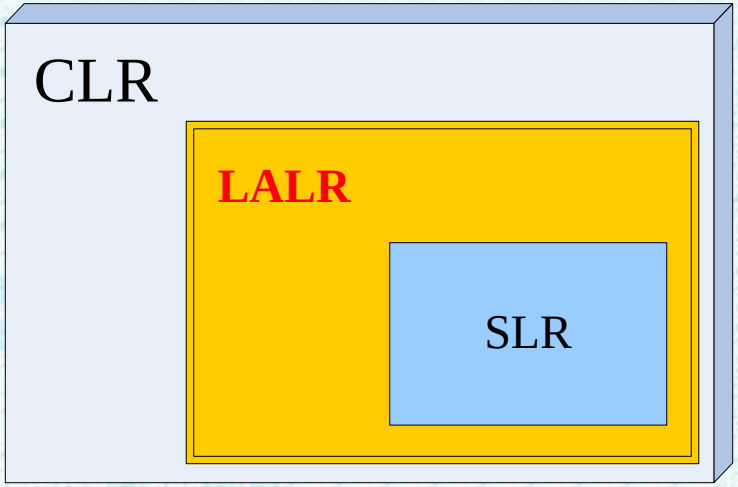
Parser Generating Systems



- The driver routine is the ***same*** for all LR parsers; only the ***parsing table*** changes from one parser to another.

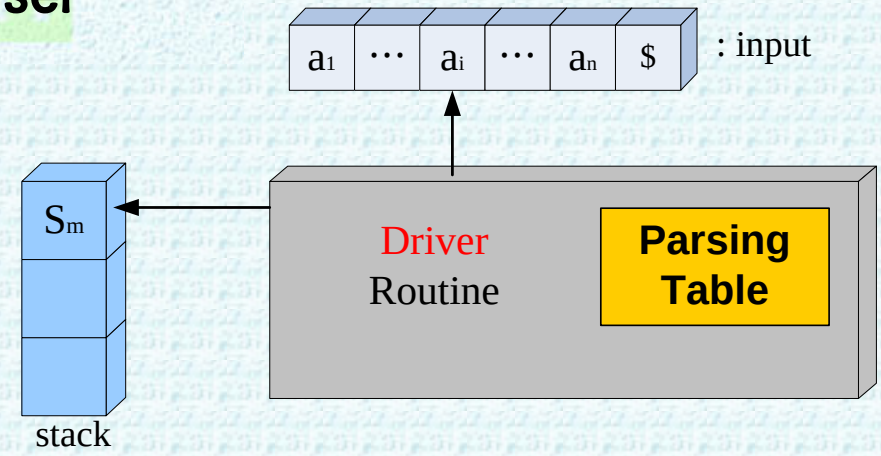
Three Methods

- The *techniques* for producing LR parsing tables
  - Simple LR(*SLR*) - LR(0) items, *FOLLOW*
  - Canonical LR(*CLR*) - LR(1) items
  - Lookahead LR(*LALR*) - ① LR(1) items
    ② LR(0), *Lookahead*



LR Parser의 구조 [1/3]

- LR parser



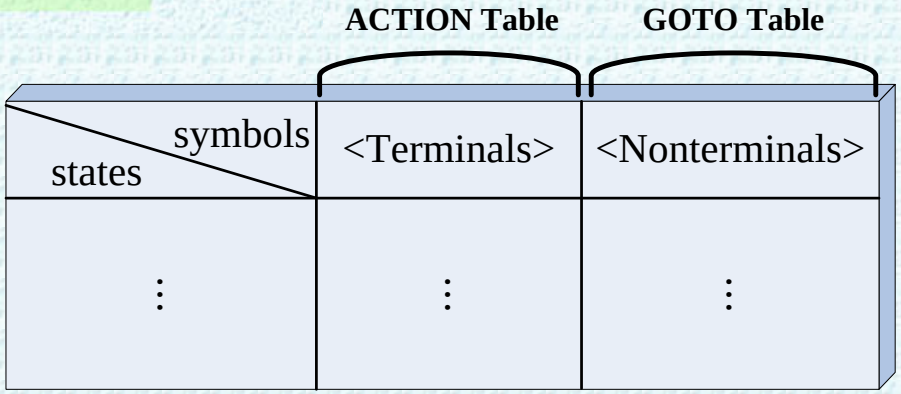
- Stack : $S_0X_1S_1X_2 \bullet\bullet\bullet X_mS_m$, where S_i : state and $X_i \in V$.
- Configuration of an LR parser :

$$(S_0X_1S_1 \bullet\bullet\bullet X_mS_m,\ a_ia_{i+1} \bullet\bullet\bullet a_n\$)$$

stack contents unscanned ***input***

LR Parser의 구조 [2/3]

- LR Parsing Table (*ACTION* table + *GOTO* table)



- The LR parsing algorithm

  ::= same as the *shift-reduce* parsing algorithm.

  - Four Actions :
    - shift
    - reduce
    - accept
    - error

LR Parser의 구조 [3/3]

1. ACTION[S_m,a_i] = ***shift*** S

   ::= ($S_0X_1S_1 \ldots X_mS_m$, $a_ia_{i+1} \ldots a_n$\$)

   $\Rightarrow$ ($S_0X_1S_1 \ldots X_mS_m$**a_iS**, $a_{i+1} \ldots a_n$\$)

2. ACTION[S_m,a_i] = ***reduce*** A $\rightarrow$ α and $|\alpha|$ = r

   ::= ($S_0X_1S_1 \ldots X_mS_m$, $a_ia_{i+1} \ldots a_n$\$)

   $\Rightarrow$ ($S_0X_1S_1 \ldots$ **$X_{m-r}S_{m-r}$**, $a_ia_{i+1} \ldots a_n$\$), GOTO(S_{m-r} , **A**) = **S**

   $\Rightarrow$ ($S_0X_1S_1 \ldots X_{m-r}S_{m-r}$**AS**, $a_ia_{i+1} \ldots a_n$\$)

3. ACTION [S_m,a_i] = ***accept***, parsing is completed.

4. ACTION [S_m,a_i] = ***error***, the parser has discovered an error and calls an error ***recovery*** routine.

LR 파싱 예제 [1/2]

- G: 1. LIST → LIST , ELEMENT
  2. LIST → ELEMENT
  3. ELEMENT → a

- Parsing Table :

states \ symbols	a	,	$	LIST	ELEMENT
0	s3			1	2
1		s4	acc		
2		r2	r2		
3		r3	r3		
4	s3				5
5		r1	r1		

where, sj means *shift* and stack state j,
ri means *reduce* by production numbered i,
acc means *accept*, and blank means *error*.

LR 파싱 예제 [2/2]

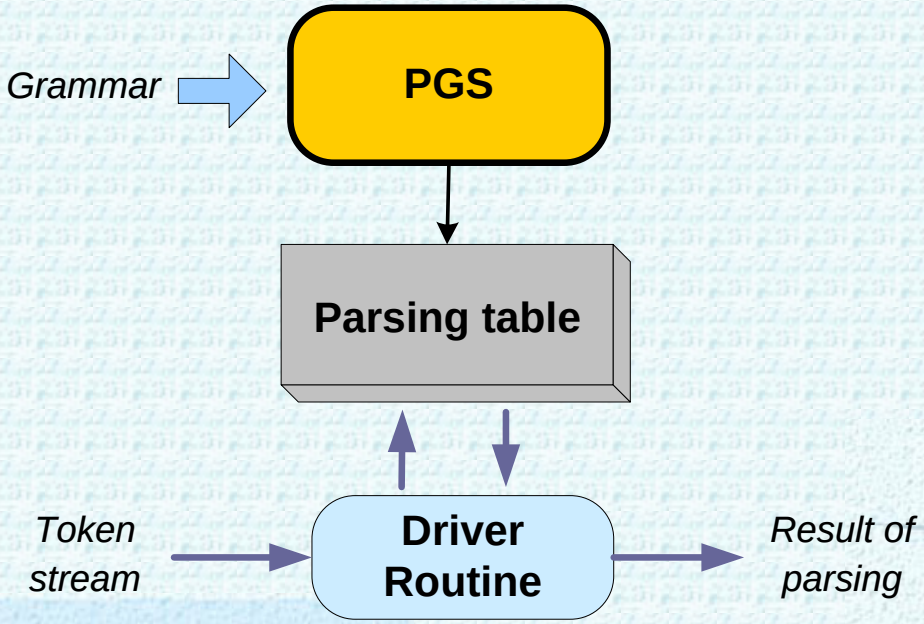
- **Input** : ω = **a, a**
- **Parsing Configuration :**

initial configuration

STACK	INPUT	ACTION	
0	a,a$	s3	
0 a 3	,a$	r3	GOTO 2
0 ELEMENT 2	,a$	r2	GOTO 1
0 LIST 1	,a$	s4	
0 LIST 1, 4	a$	s3	
0 LIST 1, 4 a 3	$	r3	GOTO 5
0 LIST 1, 4 ELEMENT 5	$	r1	GOTO 1
0 LIST 1	$	accept	

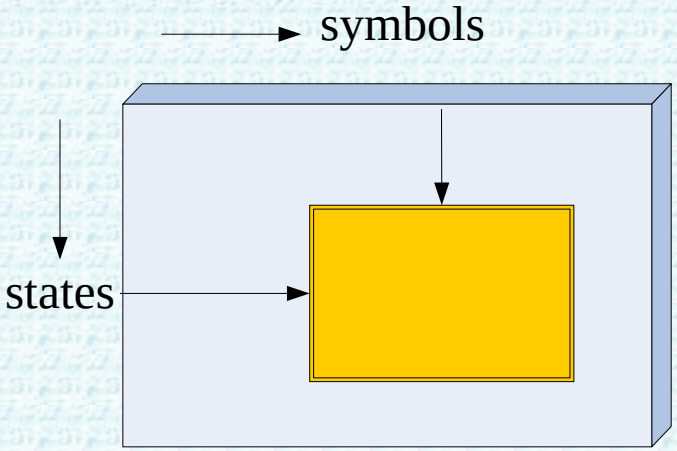
8.7 구문 분석기의 작성 [1/3]

- **Parser Generating System**



구문 분석기의 작성 [2/3]

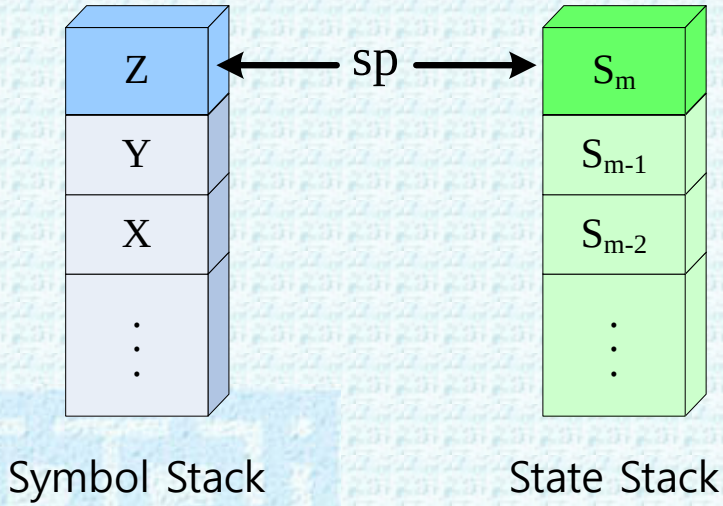
- **Parsing Table 구조**



- ptbl[S,X] = shift : > 0
  reduce : < 0
  accept : NO_RULES + 1
  error : 0

구문 분석기의 작성 [3/3]

- Parsing stack
  - Parsing stack은 병렬로 운행되는 Symbol stack과 State stack으로 구성
    - Symbol stack : 문법 심벌 저장
    - State stack : 상태 저장



LR parser for Mini C

- Text pp.361-365 에 수록

- Mini C Grammar(Text. pp. 619-622)
  - (1) number of rules : 97
  - (2) number of symbols : 85
  - (3) number of states : 153

AST를 위한 문법

Mini C grammar

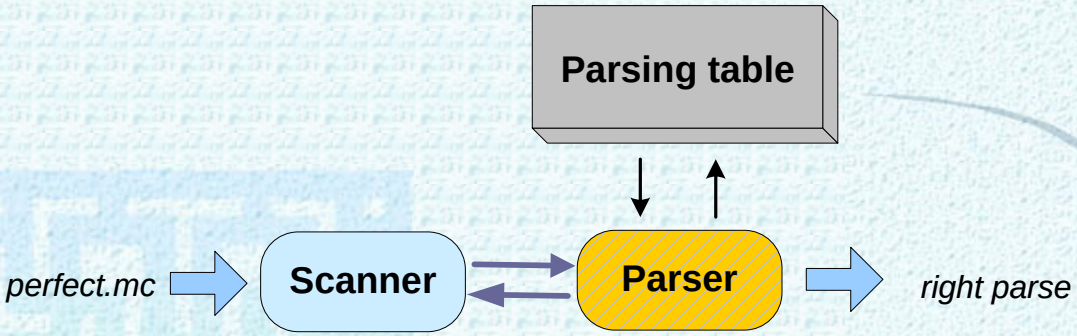
1. 부록 A : pp.619-622
   - LALR(1) 문법
   - AST를 구성하기에 적당한 형태
   - miniC.tbl
     - #define NO_RULES 97
     - #define GOAL_RULE (NO_RULES + 1)
     - #define NO_SYMBOLS 85
     - #define NO_STATES 153

2. 10.3 절 : pp.434-437
   - 부록 A에 있는 문법에 AST 정보를 추가
   - 스탠포드 PGS의 입력 형태

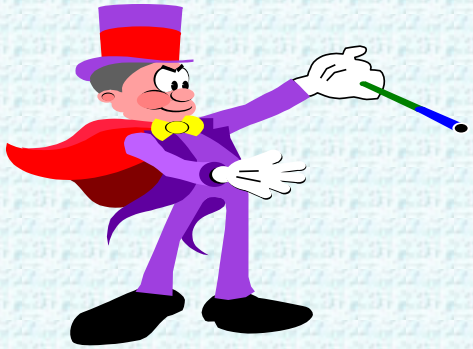
Programming Assignment #2

- Implement an LR parser for Mini C grammar(pp.371-372 => 8.20)
- Problem Specifications
  - input : perfect.mc(Text p. 616-617)
  - output : right parse
  - methods :
    - use the existing scanner for Mini C.
    - get the parsing table for Mini C at the web site.

      http://pl.skuniv.ac.kr
    - program an LR parser(Text pp.361-365)



LR Parsing



Parser Main Routine(pp.361-362)

```
#include <stdio.h>
#include "scanner.c "
#include "miniC.tbl "          // Parsing Table

#define NO_RULES               70            // number of rules
#define GOAL_RULE              (NO_RULES+1)  // accept rule
#define NO_SYMBOLS 74          // number of grammar symbols
#define NO_STATES              131           // number of states
#define PS_SIZE    100         // size of parsing stack

void dpush(int,int);
int sp,pstk[PS_SIZE];
void semantic(int);
void parser();

void main()
{
   cout<<" start of parser"<<endl;
   parser();
   cout<<" end of parser"<<endl;
}
```

Parser Main Routine

```
void parser()
{
        int entry, ruleNumber, lhs;
        int current_state;
        struct tokentype token ;

        sp = 0; pstk[sp] = 0;
        token =  scanner();
        while (1) {
                current_state = pstk[sp];
                entry = ptbl[current_state][token.number];
                if (entry > 0)              // shift action
                {  dpush(token.number, entry);
                   token = scanner();
                }
```

Parser Main Routine

```
            else if (entry < 0 )          // reduce action
            {   ruleNumber = -entry;
                if (ruleNumber == GOAL_RULE)        // accept action
                {   printf(" *** valid source ***\n");
                    return;
                }
                semantic(ruleNumber);
                sp = sp - right[ruleNumber]*2;
                lhs =  left[ruleNumber];
                current_state = ptbl[pstk[sp]][lhs];
                dpush(lhs, current_state);
            }
            else {
                printf(" === error in source ===\n");
                exit(1);
            }
        }
    }
```

Parser Main Routine

```
void semantic(int n)
{
        cout<<"reduced rule number = "<< n <<endl;
}

void dpush(int a, int b)
{
        pstk[++sp] = a;
        pstk[++sp] = b;
}
```

Mini C Program(Perfect Number)(p.616)

```
const int max = 500;
void main()
{     int  i, j, k;
      int  rem, sum ;

      i = 2;
      while (i<= max) {
                sum = 0;
                k = i / 2;
                j = 1;
                while (j <= k) {
                    rem  = i % j;
                    if (rem == 0) sum += j;
                    ++j;
                }
                if (i == sum) write(i);
                ++i;
      }
}
```

- **LALR(1) Grammar의 예제 – pp.345-348, pp.372-373**

0. S' -> S $
1. S -> L = R
2. S -> R
3. R -> L
4. L -> * R
5. L -> id

Token : = * id

Input Data : id=id $

LR Parser

```
#include <iostream.h>
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <ctype.h>

#define NO_RULES        5               // number of rules
#define GOAL_RULE       (NO_RULES+1)    // accept rule
#define NO_SYMBOLS      7               // number of grammar symbols
#define NO_STATES       10              // number of states
#define PS_SIZE         100             // size of parsing stack

void dpush(int,int);
int sp, pstk[PS_SIZE];
void semantic(int);
void parser();
```

Grammer

1. S -> L = R
2. S -> R
3. R -> L
4. L -> * R
5. L -> id
0. S'-> S $

Grammar Symbols

id * = $ L R S S'

LR Parser

Grammar

1. S -> L = R
2. S -> R
3. R -> L
4. L -> * R
5. L -> id
0. S'-> S $

```
int leftSymbol[NO_RULES + 1]  = { 7, 6, 6, 5, 4, 4};
int rightLength[NO_RULES+1]  = { 2, 3, 1, 1, 2, 1};
int parsingTable[NO_STATES][NO_SYMBOLS+1] = {
                    /* terminal + nonterminal    :      id  *  =  $  L  R  S  S'
                    /* terminal + nonterminal  #:       0   1  2  3  4  5  6  7
                    /* state 0 */                    { 5,  4,  0,  0,  2,  3,  1,  0},
                    /* state 1 */                    { 0,  0,  0, -6,  0,  0,  0,  0},
                    /* state 2 */                    { 0,  0,  6, -3,  0,  0,  0,  0},
                    /* state 3 */                    { 0,  0,  0, -2,  0,  0,  0,  0},
                    /* state 4 */                    { 5,  4,  0,  0,  8.  7,  0.  0},
                    /* state 5 */                    { 0,  0, -5, -5,  0,  0,  0, 0},
                    /* state 6 */                    { 5,  4,  0,   0,  8,  9,  0, 0},
                    /* state 7 */                    { 0,  0, -4, -4,  0,  0,  0, 0},
                    /* state 8 */                    { 0,  0, -3, -3,  0,  0,  0, 0},
                    /* state 9 */                    { 0,  0,  0, -1,  0,  0,  0, 0}
};
```

LR Parser

```
enum tsymbol{ tnull = -1 , tident, ttimes, tequal, teof};

struct tokentype{
        int number;             /* token number */
        char var;               /* token value */
};

struct tokentype token;

void scanner();     void parser();

void main()
{
        printf(" start of parser\n");
        parser();
        printf(" end of parser\n");
}
```

LR Parser

```
void scanner()
{       char ch;
        token.number = tnull;
        do {
                while (isspace(ch = getchar())) ;
                if (ch == 'a')
                {       token.number = tident;
                        token.var = ch; }
                else switch(ch)
                {       case '*': token.number = ttimes;
                                break;
                        case '=': token.number = tequal;
                                break;
                        case '$': token.number = teof;
                                break;
                        default: return;
                }
        } while (token.number == tnull);
}
```

LR Parser

```
void parser()
{
        int entry, ruleNumber, lhs;
        int current_state;
        struct tokentype token ;

        sp = 0; pstk[sp] = 0;
        token =  scanner();
        while (1) {
                current_state = pstk[sp];
                entry = parsingTable[current_state][token.number];
                if (entry > 0)              // shift action
                {  dpush(token.number, entry);
                   token = scanner();
                }
```

LR Parser

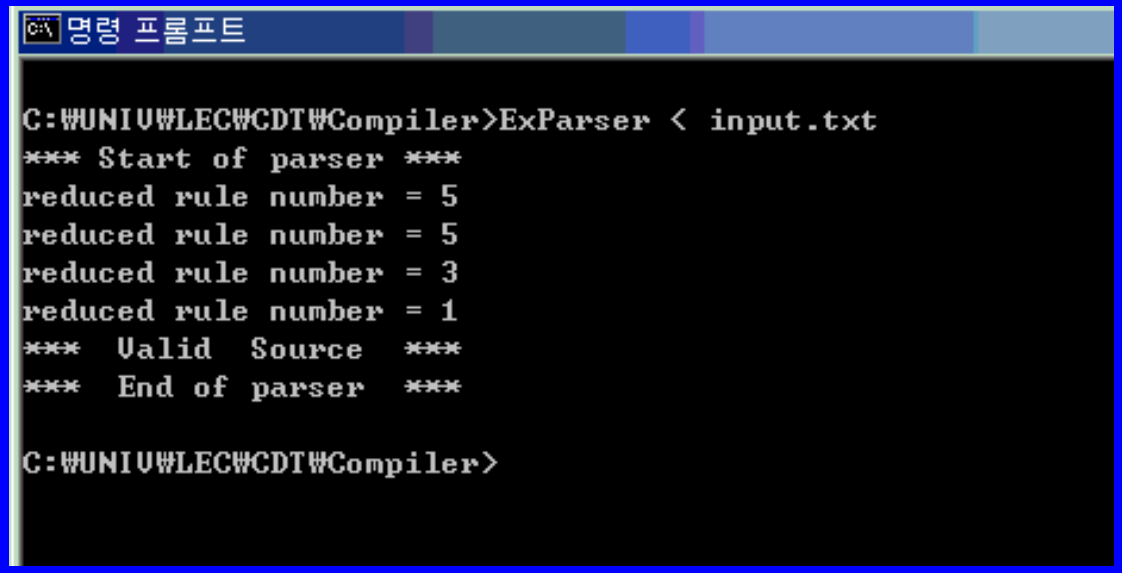
```
            else if (entry < 0 )          // reduce action
            {   ruleNumber = -entry;
                if (ruleNumber == GOAL_RULE)     // accept action
                {   printf(" *** valid source ***\n");
                    return;
                }
                semantic(ruleNumber);
                sp = sp - rightLength[ruleNumber]*2;
                lhs =  leftSymbol[ruleNumber];
                current_state = parsingTable[pstk[sp]][lhs];
                dpush(lhs, current_state);
            }
            else {
                printf(" === error in source ===\n");
                exit(1);
            }
        }
    }
```

LR Parser

```
void semantic(int n)
{
        printf("reduced rule number = %d\n", n);
}

void dpush(int a, int b)
{
        pstk[++sp] = a;
        pstk[++sp] = b;
}
```

Data : a=a$


명령 프롬프트
C:₩UNIV₩LEC₩CDT₩Compiler>ExParser < input.txt
*** Start of parser ***
reduced rule number = 5
reduced rule number = 5
reduced rule number = 3
reduced rule number = 1
*** Valid Source ***
*** End of parser ***
C:₩UNIV₩LEC₩CDT₩Compiler>