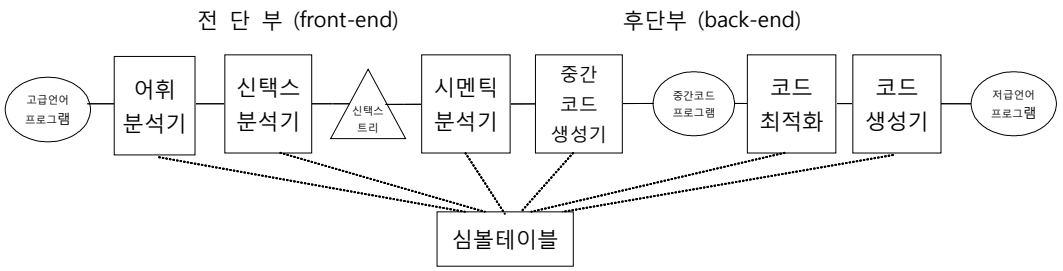
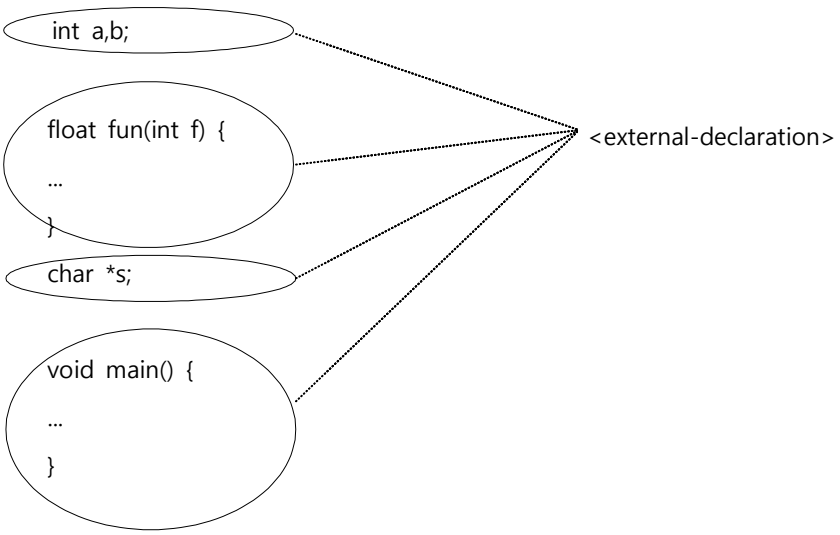
1 장 간단한 번역기

1.1 컴파일러의 구조



1.2 문법과 언어



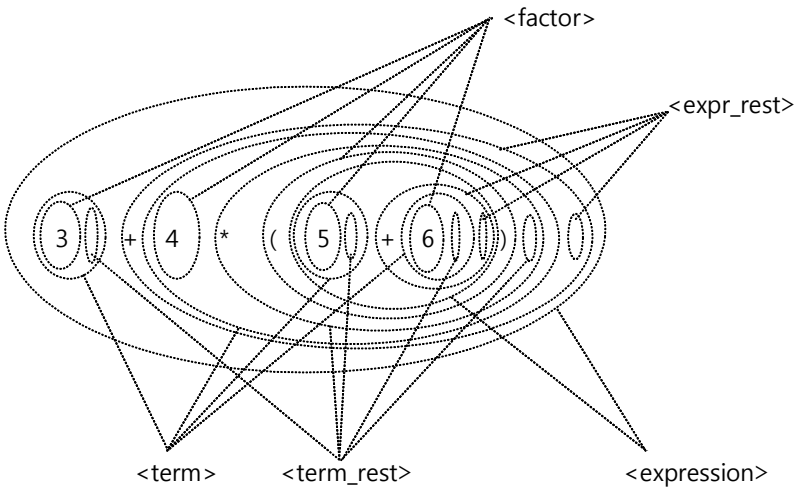
어휘 (vocabulary), 단어 (*word*), *심볼* (*symbol*)
센텐스 (*sentence, 문장*)
문법 (*grammar*), *신택스* (*syntax*)
생성 규칙 (*production*)
시멘틱스 (*semantics*)

간단한 형태의 수식을 위한 문법

G_0 :

```
<expression>   ::=  <term>  <expr_rest>
<expr_rest>    ::=  + <term> <expr_rest>
                |   λ
<term>         ::=  <factor>  <term_rest>
<term_rest>    ::=  * <factor>  <term_rest>
                |   λ
<factor>       ::=  <number>
                |   (  <expression>  )
<number>       ::=  0  |  1  |  …  |  9
```

"3+4*(5+6)"의 구조



Context-Free Grammar(문맥자유문법) G = (T, N, P, S)

- T: terminal symbols
- N: nonterminal symbols
- P: production rules ‘A ::= α’ , A$\in$N , $\alpha\in(N\cup T)^*$
- S: start symbol, S$\in$N

G_1 = ({+,*,(,),0,1,2,3,4,5,6,7,8,9}, {E,R,T,Q,F,N}, P_1, E)

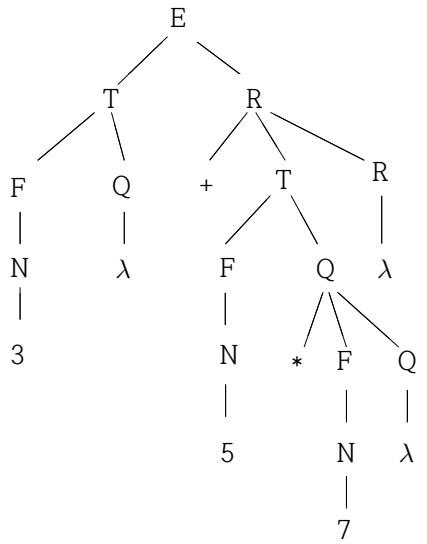
P_1:
```
E  ::=  T  R
R  ::=  + T R
     |  λ
T  ::=  F Q
Q  ::=  * F Q
     |  λ
F  ::=  N
     |  ( E )
N  ::=  0  |  1  |  ...  |  9
```

$A \Rightarrow \alpha$, $A \Rightarrow^{*} \alpha_n$, $A \Rightarrow^{+} \alpha_n$

derivation

$E \Rightarrow T\ R \Rightarrow F\ Q\ R \Rightarrow N\ Q\ R \Rightarrow 3\ Q\ R \Rightarrow 3\ R \Rightarrow 3 + T\ R$
$\Rightarrow 3 + F\ Q\ R \Rightarrow 3 + N\ Q\ R \Rightarrow 3 + 5\ Q\ R \Rightarrow 3 + 5 * F\ Q\ R$
$\Rightarrow 3 + 5 * N\ Q\ R \Rightarrow 3 + 5 * 7\ Q\ R \Rightarrow 3 + 5 * 7\ R \Rightarrow 3 + 5 * 7$

syntax tree



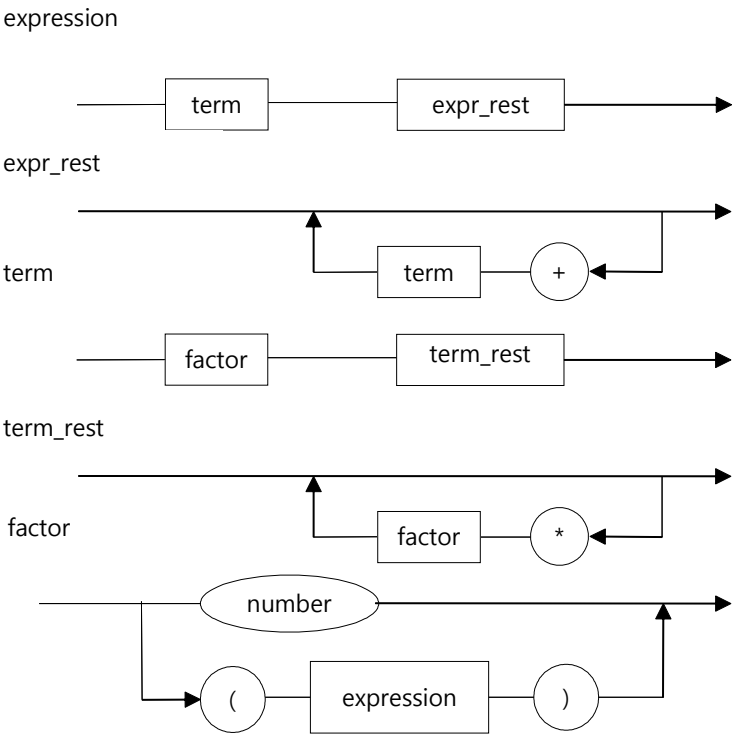
sentence

x is a sentence if $S \Rightarrow^{*} x$, $x \in T^{*}$

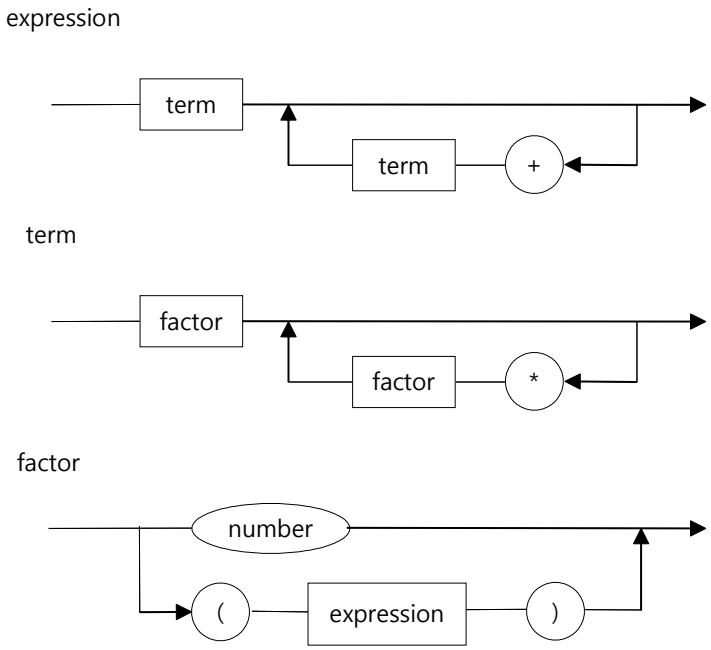
language

$L(G) = \{\ x \in T^{*} \mid S \Rightarrow^{*} x\ \}$

syntax graph

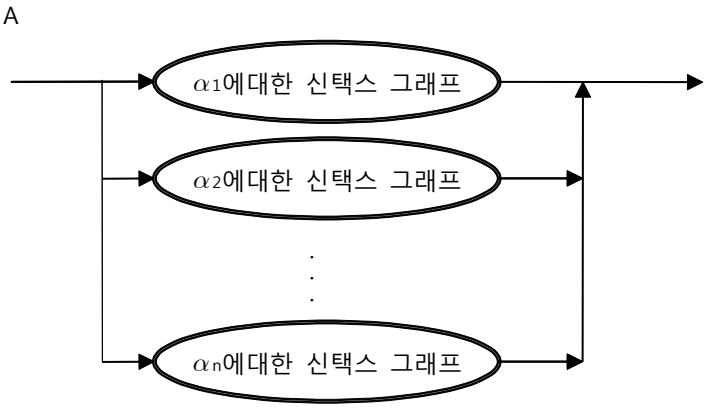


syntax graph for simple expressions

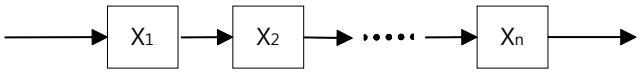


t문맥자유문법에서 신택스 그래프로 변환

(규칙-1) A ::= α_1 | α_2 | ... |α_n

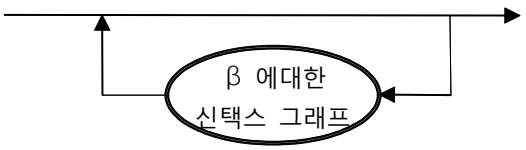


(규칙-2) 각 α 가 $X_1 X_2 \ldots X_m$ 이면
각 α 에 대한 신택스 그래프:



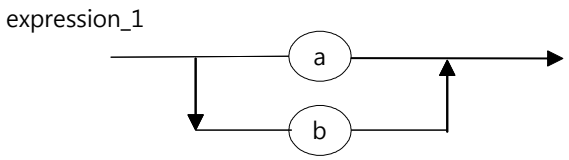
각 심볼 X 가 넌터미날 심볼인 경우는 사각형, 터미날 심볼인 경우는 동그라미

(규칙-3) 특정 패턴 β 를 반복하는 경우, 즉 β 가 0번 이상 반복하는 형태:



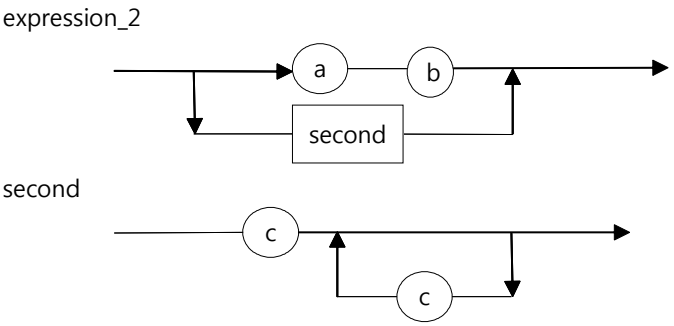
1.3 신택스 그래프와 재귀하강식 (Recursive-Descent) 파서

'a' 나 'b' 같은 스트링만 인식




```
char ch;
void expression_1( ) {
        if (ch=='a')
                ch=getchar( );
        else if (ch=='b')
                ch=getchar( );
        else
                error();
}
void main() {
        ch=getchar( );
        expression_1( );
        if (ch!=EOF)
                error();
}
```

'ab' 나 'c...c'를 인식




```
char ch=' ';
char get_token( ) {
        while (ch==' ')
            ch=getchar( );
        return (ch);
}
void expression_2( ) {
        if (ch=='a')
            ch=get_token( );
            if (ch=='b')
                ch=get_token( );
            else
                error( );
        else
            second( );
}
```

```
void second ( ) {
        if (ch=='c') {
            ch=get_token( );
            while (ch=='c')
                ch=get_token( ); }
        else
            error( );
}

void main( ) {
        ch=get_token( );
        expression_2( );
        if (ch!=EOF)
            error( );
}
```

1.6의 신택스 그래프를 위한 파서 프로그램

```
enum {NULL, PLUS, STAR, NUMBER,
        LPAREN, RPAREN, END} token;

void get_token ( ) {
   // next token of input --> token
}

void expression ( ) {
      term();
      while (token==PLUS) {
         get_token( );
         term( ); }
}

void term ( ) {
      factor();
      while (token==STAR) {
         get_token( );
         factor( ); }
}

void factor ( ) {
      if (token==NUMBER)
         get_token( );
      else if (token==LPAREN) {
         get_token( );
         expression( );
         if (token==RPAREN)
                  get_token( );
         else
                  error( ); }
      else error();
}
main ( ) {
      get_token( );
      expression( );
      if (token!=END)
         error( );
}
error ( ) {
      // error handling
}
```

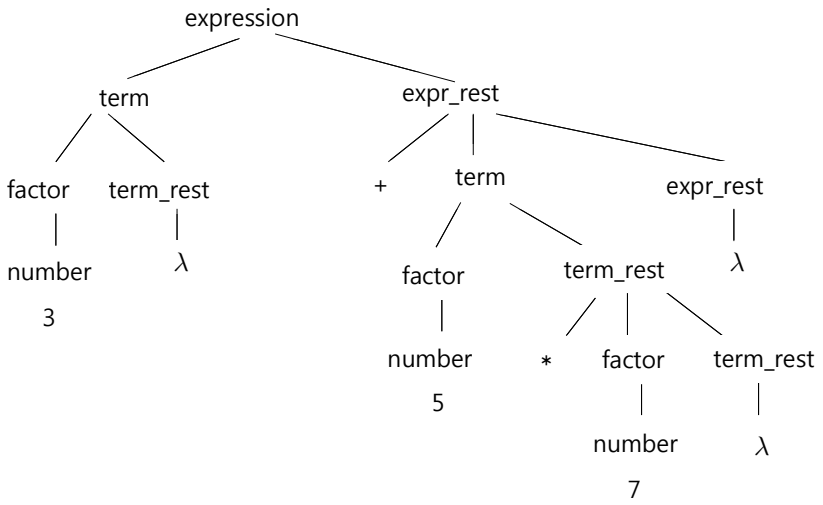
get_token() 함수 프로그램 (어휘 분석기 혹은 스캐너)

입력 문자	토큰 종류 (token)
'0'~'9'	NUMBER
'+'	PLUS
'*'	STAR
'('	LPAREN
')'	RPAREL
EOF	END
기타	NULL

우리 수업중 약속이라고 생각,
앞으로 코딩할때 규칙따를것

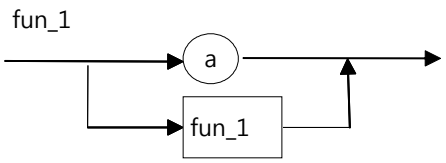
재귀하강적 파서 (recursive-descent parser) 의 작동

main() --> expression()--> term() --> factor() --> expression()
시작기호로 부터 하향식 호출, 좌측 먼저 처리
한개씩의 토큰만 필요



신택스 그래프의 제한사항

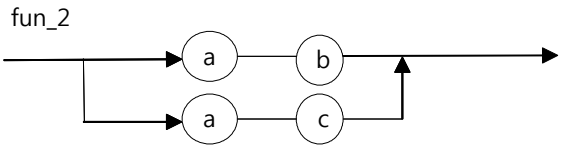
<경우-1>



else로 빠질때,
무한룹돌게됨

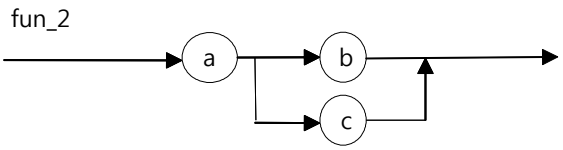
```
void fun_1 ( ) {
    if (token=='a')
        get_token( );
    else
        fun_1( );
}
```

<경우-2> “ab” 나 “ac” 인식

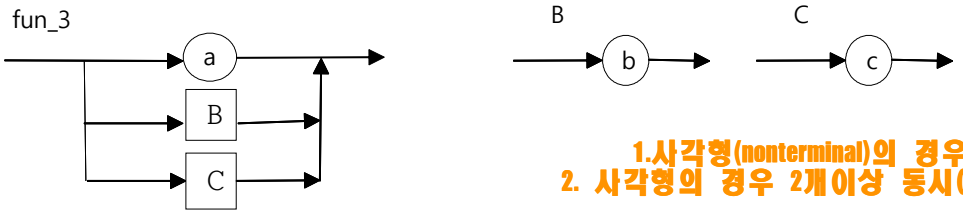



```
void fun_2 ( ) {
    if (token=='a') {
        get_token( );
        if (token=='b')
                get_token( );
        else    error( ); }
    else if (token=='a') {
        get_token( );
        if (token=='c')
                get_token( );
        else    error( ); }
    else   error( );
}
```

수정한 신택스 그래프



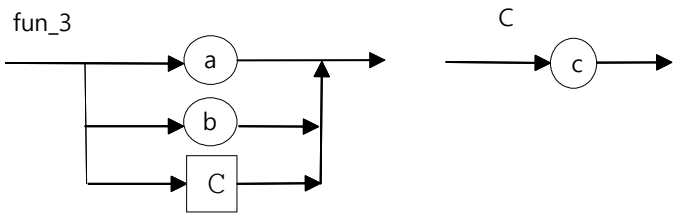
<경우-3> “a” 나 “b” 나 “c” 인식



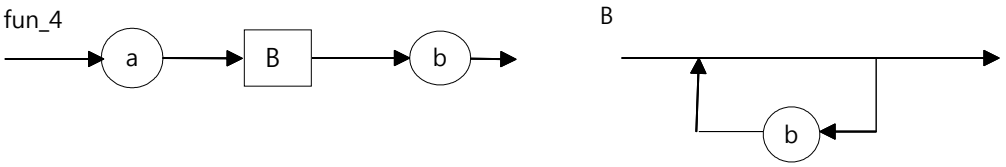
1.사각형(nonterminal)의 경우 맨위에 올 수 없다.
2. 사각형의 경우 2개이상 동시에 같은 level에 올 수 없다

```
void fun_3 ( ) {
    if (token=='a')
        get_token( );
    else
        B( );
    else                          // 작성 오류
        C( );
    .....
}
void B( ) {
    if (token=='b')
        get_token( );
    else
        error( );
}
void C( ) {
    // 생략
}
```

수정한 신택스 그래프



<경우-4> ‘ab….b’




```
void fun_4( ) {
    if (token=='a')
        get_token( );
        B( );
                    if (token=='b')
                    get_token( );
        else
                    error( );
    else
        error( );
}
void B( ) {
    while (token=='b') {
        get_token( );
}
```

1.4 번역기와 수식 계산

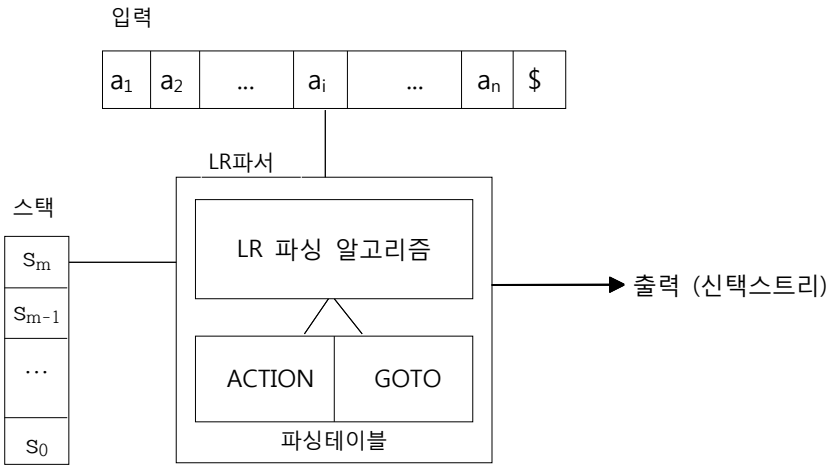
```
int num;
enum {NULL,NUMBER,PLUS,STAR,
        LPAREN,RPAREN,END} token;
void main ( ) {
        int result;
        get_token();
        result=expression();
        if (token!=END)
            error(3);
        else
            printf("%d ₩n",result);
}

int expression ( ) {
        int result;
        result=term();
        while (token==PLUS) {
            get_token();
            result=result+term( ); }
        return (result);
}
int term ( ) {
        int result;
        result=factor();
        while (token==STAR) {
            get_token();
            result=result*factor( ); }
        return (result);
}
```

```
int factor ( ) {
        int result;
        if (token==NUMBER) {
            result=num
            get_token(); }
        else if (token==LPAREN) {
            get_token();
            result=expression();
            if (token==RPAREN)
                get_token();
                else
                    error(2); }
        else
            error(1);
        return (result);
}
void get_token ( ) {
        // next token     --> token
        // number value --> num
}
void error (int i) {
        switch (i) {
            case 1: ... break;
            case 2: ... break;
            case 3: ... break;
        }
        exit(1);
}
```

해당 위치에 print를 찍으면
infix를 postfix로 표현 할 수 있다

1.5 상향식 (Bottom-up) 신택스 분석과 LR 파서

LR 파서 구조



상태번호
스택 : s_0 … s_m
끝표시 심볼 (*end-marker*) '$'
파싱테이블: ACTION[], GOTO[]

총 4가지

파싱 동작:

- "shift i"
- "reduce j"
- "accept"
- "error"

파서상황 (*configuration*)

$Q : (S_0 X_1S_1 \ldots X_mS_m, \quad a_i \ldots a_n\$)$

초기 파서상황

$Q_0 : (0, \quad x \$), \quad x=a_0 \ldots a_n$

스택 꼭대기: S_m , 입력 토큰이 a_i 인 경우, ACTION[S_m][a_i]을 참조

- "shift i" 인 경우:
  ($S_0 X_1S_1 \ldots X_mS_m, a_ia_{i+1}\ldots a_n\$$) 를 ($S_0 X_1S_1 \ldots X_mS_m\ a_i\ i,\ a_{i+1}\ldots a_n\$$) 로 변경
- "reduce j" 인 경우:
  문법의 j 번째 규칙: 'A ::= α' , r=|α|,
  ($S_0 X_1S_1 \ldots X_mS_m, a_i\ldots a_n\$$) 를 ($S_0 X_1S_1 \ldots X_{m-r}S_{m-r}, a_i\ldots a_n\$$) 로 변경
  GOTO[s_m][A]의 값이 k 면 파서상황을 ($S_0 X_1S_1 \ldots X_{m-r}S_{m-r}$ Ak, $a_i\ldots a_n\$$) 로 변경
- "accept" 인 경우
  성공적 종료
- "error" 인 경우:
  에러 처리

스택 top과 인풋토큰사이의 관계를
parsing table에서 구해서 결정

<LR 파싱 알고리즘>

```
push(0)
token=get_token()
while (1) {
        stack top의 상태번호를 s 라고 가정한다
        switch (ACTION[s][token]) {
                case "shift i" :
                        push(token)
                        push(i)
                        token=get_token()
                        break
                case "reduce j" :
                        j번째 생성규칙을 "A -> α" 라고 가정한다
                        스택에서 2|α| 만큼의 기호를 삭제한다
                        삭제후 새로운 stack top 의 상태번호를 t 라고 가정한다
                        push(A)
                        push(GOTO[t][A])
                        break
                case "accept" :
                        파서 종료처리
                default :
                        error() 함수호출
}
```

로 수정, 화살표에서 가로두개짜리는 유도(deriviation)임
하나짜리를 사용해야함

Si 〉 입력토큰하나를 스택에 넣고 i번째 스테이트도 스택에 넣기
Ri 〉 i번 규칙으로 reduce
〉〉i번 규칙에 해당하는 right hand side를 left hand side로 바꾸고
해당하는 goto테이블을 참고해서 번호쓰기

① E → E + T
② | T
③ T → T * F
④ | F
⑤ F → (E)
⑥ | n

테이블 / 상태번호 / 심볼	ACTION 테이블						GOTO 테이블		
	n	+	*	(	)	$	E	T	F
0	S5			S4			1	2	3
1		S6				acc			
2		R2	S7		R2	R2			
3		R4	R4		R4	R4			
4	S5			S4			8	2	3
5		R6	R6		R6	R6			
6	S5			S4				9	3
7	S5			S4					10
8		S6			s11				
9		R1	S7		R1	R1			
10		R3	R3		R3	R3			
11		R5	R5		R5	R5			

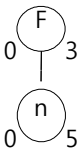
텍스트

	스 택	입 력
	(0,	n + n * n $)
S5 ↦	(0 n 5,	+ n * n $)
R6 ↦	(0 F 3,	+ n * n $)
R4 ↦	(0 T 2,	+ n * n $)
R2 ↦	(0 E 1,	+ n * n $)
S6 ↦	(0 E 1 + 6,	n * n $)
S5 ↦	(0 E 1 + 6 n 5,	* n $)
R6 ↦	(0 E 1 + 6 F 3,	* n $)
R4 ↦	(0 E 1 + 6 T 9,	* n $)
S7 ↦	(0 E 1 + 6 T 9 * 7,	n $)
S5 ↦	(0 E 1 + 6 T 9 * 7 n 5,	$)
R6 ↦	(0 E 1 + 6 T 9 * 7 F 10,	$)
R3 ↦	(0 E 1 + 6 T 9,	$)
R1 ↦	(0 E 1,	$)
acc		

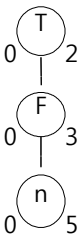
0 , n + n * n $

S5 ↦ 

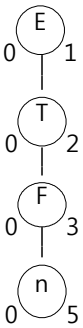
, + n * n $

R6 ↦ 

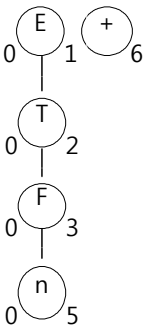
, + n * n $

R4 ↦ 

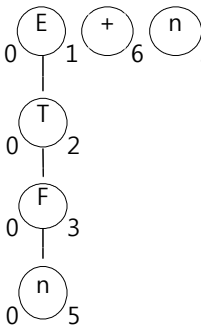
, + n * n $

R2 ↦ 

, + n * n $

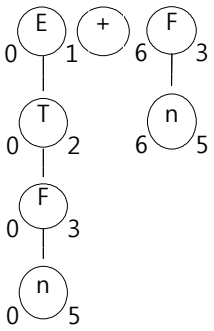
S6 ↦ 

, n * n $

S5 ↦ 

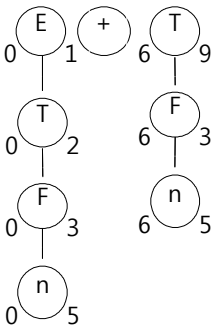
, * n $

R6 ↦



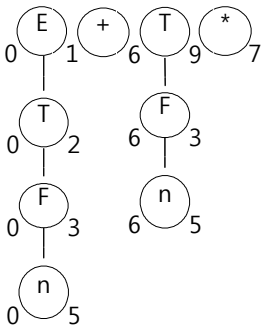
, * n $

R4 ↦



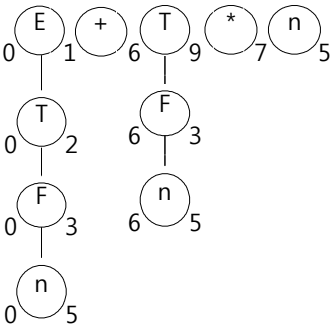
, * n $

S7 ↦



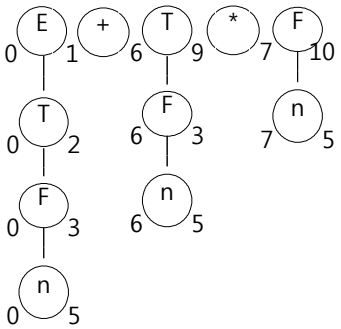
, n $

S5 ↦



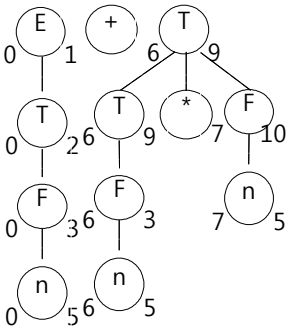
, $

R6 ↦



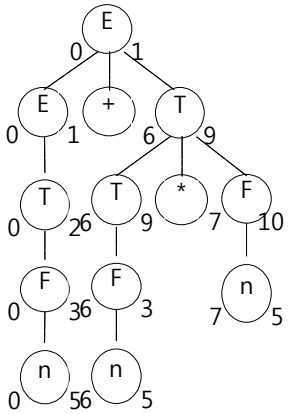
, $

R3 ↦



, $

R1 ↦



, $ acc ↦

syntax analyzing 하기 위해 필요한것
1. 해당되는 문법
2. 그에 맞는 parsing table

SLR, LALR, 및 LR 파서 : 파싱 알고리즘은 동일
LR 파싱 테이블
문법으로부터 제작
파싱 테이블 크기: ‘12줄*심볼갯수‘
C 언어: 500*500 정도 이상

실제로 구현하게 될 경우
input토큰을 넣을 필요없이
state 번호만 넣어놓고
s면 push
r이면 규칙에 맞는 갯수만큼지우고
goto테이블 참조해서 넣으면 끝

```
#define NUMBER 256
#define PLUS 257
#define STAR 258
#define LPAREN 259
#define RPAREN 260
#define END 261
#define EXPRESSION 0
#define TERM 1
#define FACTOR 2
#define ACC 999

int action[12][6]={
        {5, 0, 0, 4, 0, 0}, {0, 6, 0, 0, 0, ACC},{0,-2, 7, 0,-2,-2},
        {0,-4,-4, 0,-4,-4}, {5, 0, 0, 4, 0, 0},    {0,-6,-6, 0,-6,-6},
        {5, 0, 0, 4, 0, 0}, {5, 0, 0, 4, 0, 0},   {0, 6, 0, 0,11, 0},
        {0,-1, 7, 0,-1,-1}, {0,-3, -3, 0,-3,-3},   {0,-5,-5, 0,-5,-5} };

int go_to[12][3]={
        {1,2,3},{0,0,0}, {0,0,0},{0,0,0},{8,2,3},{0,0,0},
        {0,9,3},{0,0,10},{0,0,0},{0,0,0},{0,0,0},{0,0,0} };

int prod_left[7]={0,EXPRESSION,EXPRESSION,TERM,TERM,FACTOR,FACTOR};
int prod_length[7]={0,3,1,3,1,3,1};

int stack[1000];
int top=-1;
int sym;

void push(int);
void reduce(int);
void yyerror();
void lex_error();

void main () {
        yyparse();
}

int yyparse() {
        int i;
        stack[++top]=0;                         // initial state
        sym=yylex();
        do {
```

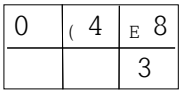
양의 정수 〉 Shift
음의 정수 〉 Reduce

```
                    i=action[stack[top]][sym-256];      // get relation
                    if (i==ACC)
                            printf("success !₩n");
                    else if (i>0)                       // shift
                            shift(i);
                    else if (i<0)                       // reduce
                            reduce(-i);
                    else
                            yyerror(); }

            while (i!=ACC);
}

void push(int I) {
            top++;
            stack[top]=i;
}

void shift(int I) {
            push(i);
            sym=yylex();
}

void reduce(int I) {
            int old_top;
            top-=prod_length[i];
            old_top=top;
            push(go_to[stack[old_top]][prod_left[i]]);
}

void yyerror() {
            printf("syntax error₩n");
            exit(1);
}

int yylex() {
            static char ch=' ';
            int i=0;
            while (ch==' '||ch=='₩t'||ch=='₩n') ch=getchar();
            if (isdigit(ch)) {
                    do
                            ch=getchar();
```

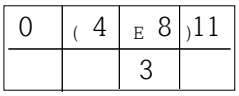
```
                while (isdigit(ch));
                return(NUMBER); }
        else if (ch=='+'){
                ch=getchar();
                return(PLUS);}
        else if (ch=='*'){
                ch=getchar();
                return(STAR);}
        else if (ch=='('){
                ch=getchar();
                return(LPAREN);}
        else if (ch==')'){
                ch=getchar();
                return(RPAREN);}
        else if (ch==EOF)
                return(END);
        else
                lex_error();
}

void lex_error() {
        printf("illegal token₩n");
        exit(1);
}
```

수식의 값을 계산, 2-트랙 스택이용

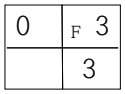
	stack	input
트랙-1 / 트랙-2	0 /	, (1 + 2) * 3 $
S4 ↦	0 (4 /	, 1 + 2) * 3 $
S5 ↦	0 (4 n 5 / 1	, + 2) * 3 $
R6 ↦	0 (4 F 3 / 1	, + 2) * 3 $
R4 ↦	0 (4 T 2 / 1	, + 2) * 3 $
R2 ↦	0 (4 E 8 / 1	, + 2) * 3 $
S6 ↦	0 (4 E 8 + 6 / 1	, 2) * 3 $
S5 ↦	0 (4 E 8 + 6 n 5 / 1 2	,) * 3 $
R6 ↦	0 (4 E 8 + 6 F 3 / 1 2	,) * 3 $
R4 ↦	0 (4 E 8 + 6 T 9 / 1 2	,) * 3 $

R1 ↦ 

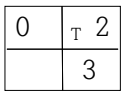
,) * 3 $

S11↦ 

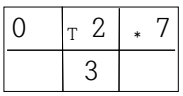
, * 3 $

R5 ↦ 

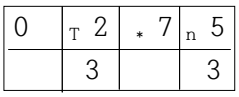
, * 3 $

R4 ↦ 

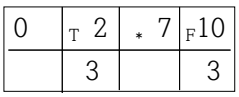
, * 3 $

S7 ↦ 

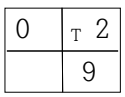
, 3 $

S5 ↦ 

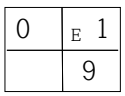
, $

R6 ↦ 

, $

R3 ↦ 

, $

R2 ↦ 

, $

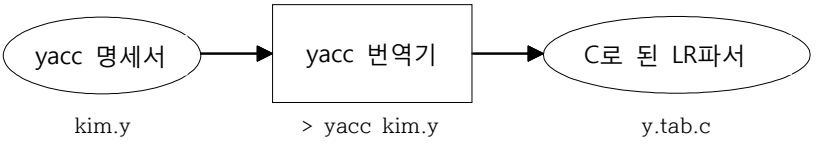
acc

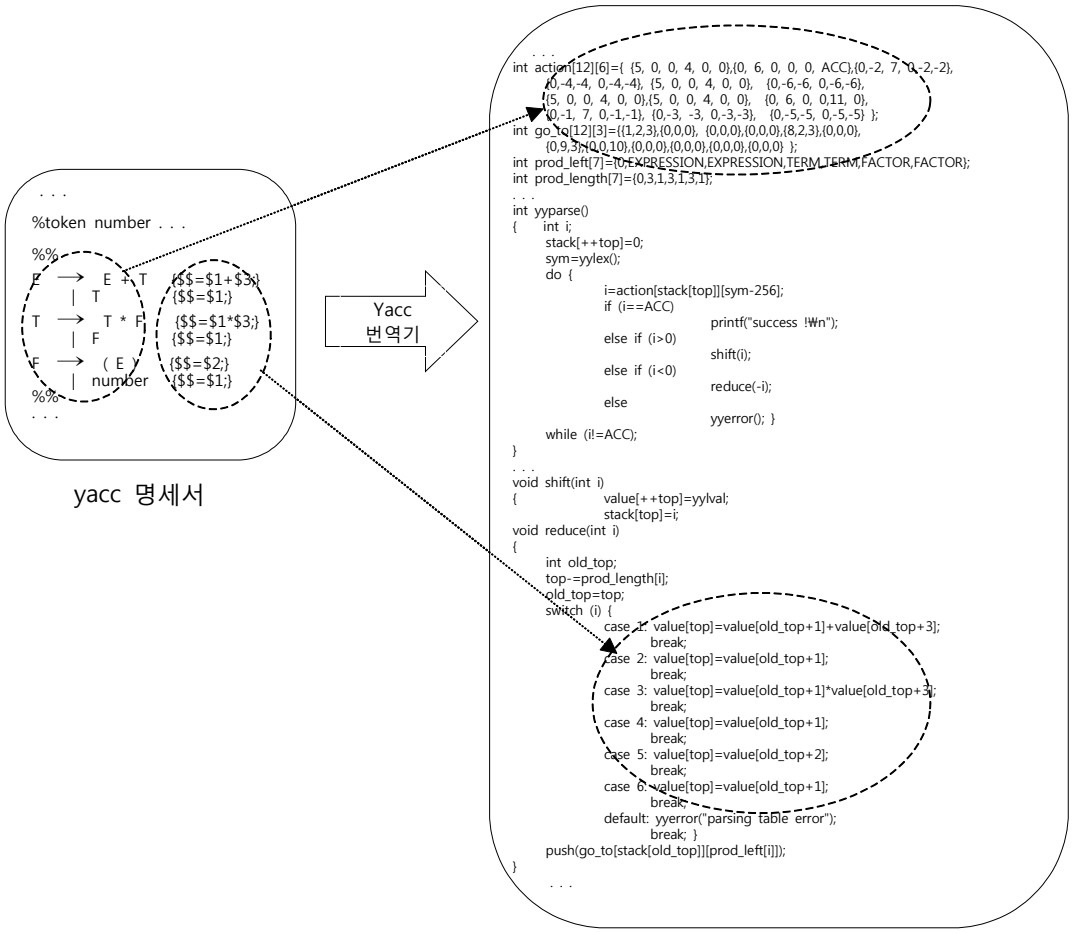
```
char yytext[32];
int yylval;
... ...
void shift(int I) {
        push(i);
        value[top]=yylval;
        sym=yylex();
}

void reduce(int I) {
        // 생략
        switch (i) {  // 규칙번호에 따른 수식 값 계산
                case 1: value[top]=value[old_top+1]+value[old_top+3];
                        break;
                case 2: value[top]=value[old_top+1];
                        break;
                case 3: value[top]=value[old_top+1]*value[old_top+3];
                        break;
                case 4: value[top]=value[old_top+1];
                        break;
                case 5: value[top]=value[old_top+2];
                        break;
                case 6: value[top]=value[old_top+1];
                        break;
                default: yyerror("parsing table error");
                        break;
        }
}

int yylex() {
        // 생략
        int i=0;
        if (isdigit(ch)) {
                do {
                        yytext[i]=ch;
                        ch=getchar();
                } while (isdigit(ch));
                yytext[i]=0;
                yylval=atoi(yytext);
                return(NUMBER); }
        // 생략
}
```

실제 값을 계산하려 할때는,
읽을때 해당하는 값을 아래 스택에 저장하고
reduce가 일어날때 마다, 계산해서 업데이트하는식

하나 혹은 두칸씩 옆으로 옮기는 과정

1.6 Yacc 번역기와 LR 파서의 생성





Yacc 명세서

선언부 (delcarations)
%%
문법과 번역규칙부 (rules)
%%
보조 프로그램부 (supporting programs)

문법 G_2

```
E  →  E + T
    |  T
T  →  T * F
    |  F
F  →  ( E )
    |  0
       ...
    |  9
```

```
%start E
%token  number

%%

E       : E '+' T
        | T
        ;
T       : T '*' F
        |  F
        ;
F       : '(' E ')'
        | '0'
          ...
        | '9'
        ;
%%

yylex( ) {
    char ch;
    ch=getchar();
    return ch;
}
void main ( ) {
    yyparse();
}
```

수식의 값을 계산, yylval 변수 이용

```
%{
#include 'y.tab.h'
%}
%start E
%token  number

%%
S       :  E            { printf("%d",$1); }
        ;
E       :  E PLUS T     { $$ = $1 + $3; }
        |  T            { $$ = $1; }
        ;
T       :  T STAR F     { $$ = $1 * $3; }
        |  F            { $$ = $1; }
        ;
F       :  LP E RP      { $$ = $2; }
        |  number       { $$ = $1; }
        ;
%%

yylex() {
   //  '+'  :    return (PLUS);
   //  '*'  :    return (STAR);
   //  '('  :    return (LP);
   //  ')'  :    return (RP);
   //  정수 :    yylval=정수값;  return (number);
}
void main () {
   yyparse();
}
```

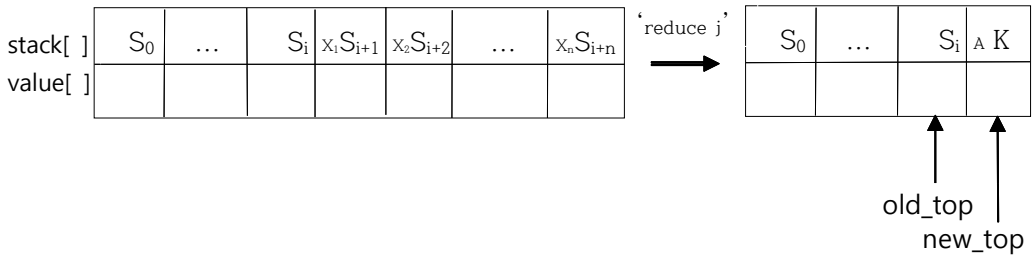
```
#define PLUS    1
#define STAR    2
#define LP      3
#define RP      4
#define number  5
```

action rules

번역규칙: A ::= α_1 | α_2 | ... |α_n

A : α_1 {번역규칙$_1$}
| α_2 {번역규칙$_2$}
| ... ...
| α_n {번역규칙$_n$}

'reduce' 동작



$$, $1, $2, $3 의 의미