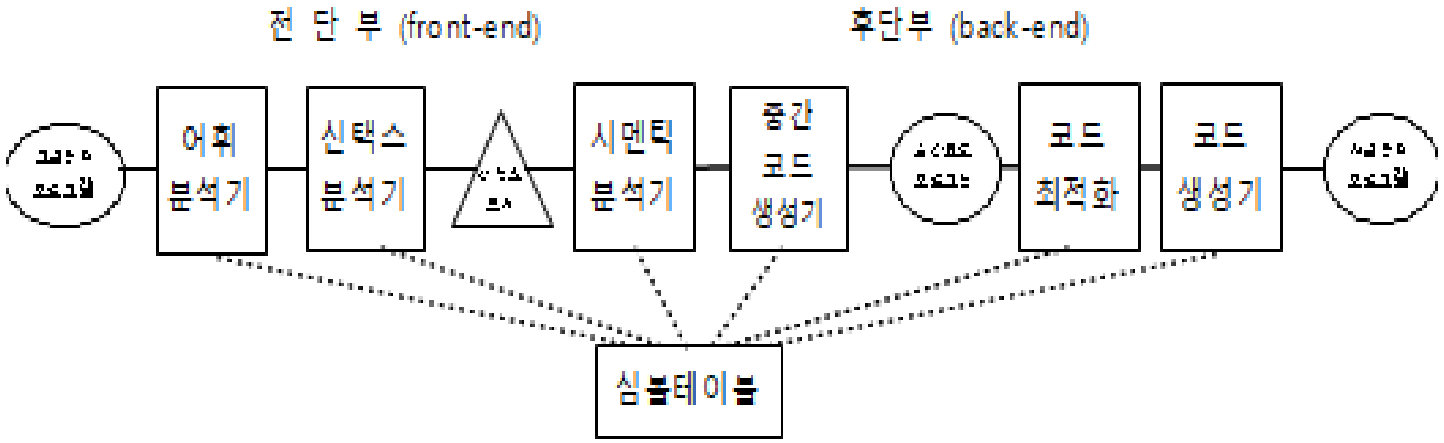
C 언어와 컴파일러

소프트웨어 학부

2023 봄학기

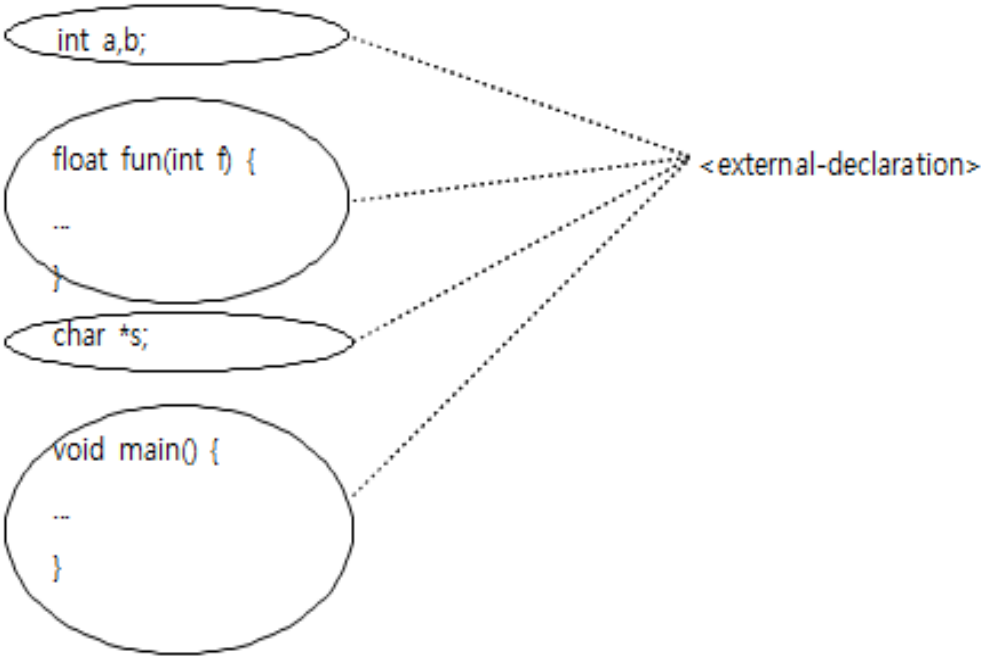
유재우 교수

컴파일러의 구조


전 단 부 (front-end)
후단부 (back-end)
어휘
분석기
신택스
분석기
시멘틱
분석기
중간
코드
생성기
코드
최적화
코드
생성기
심볼테이블

문법과 언어

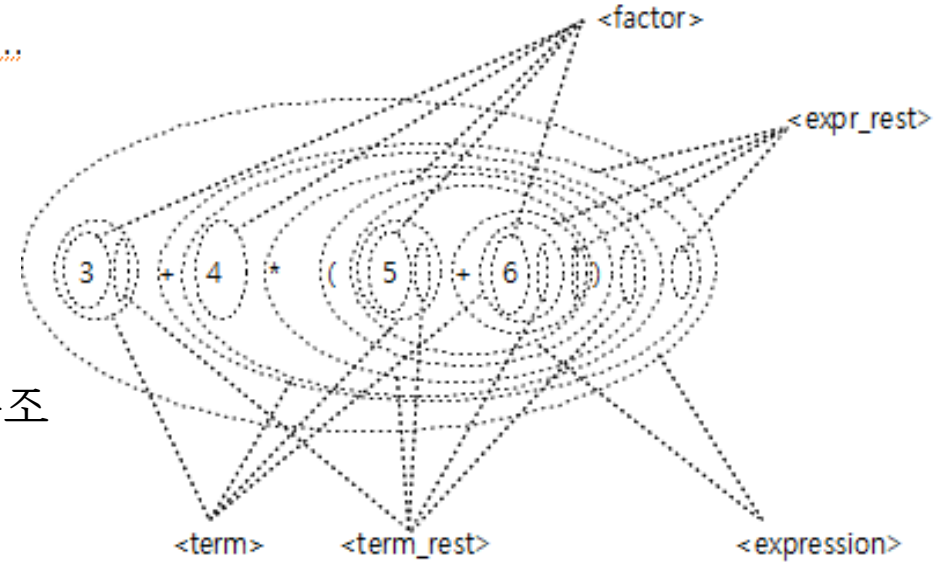
- Grammar
- Syntax, Semantics
- Language



간단한 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 | ...



수식 “3+4*(5+6)”의 문법적구조

Context-Free Grammar

- 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:

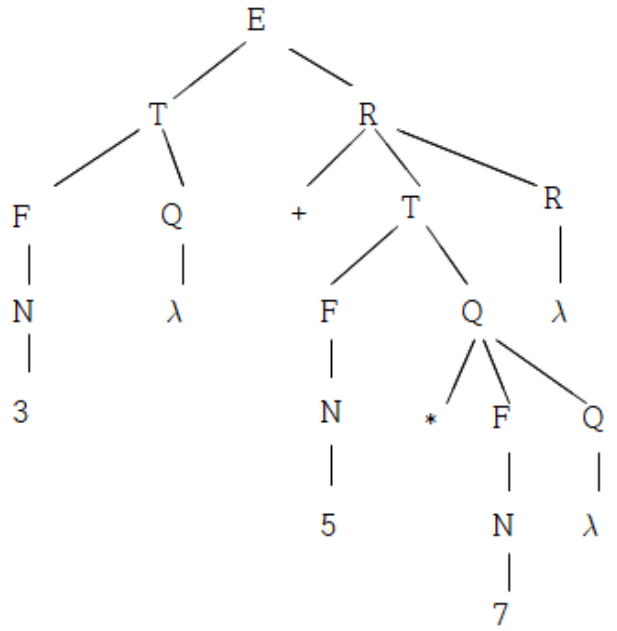
E ::= T R
R ::= + T R
  | λ
T ::= F Q
Q ::= * F Q
  | λ
F ::= N
  | (E)
N ::= 0 | 1 | ... | 9

Sentence & Language of a Grammar

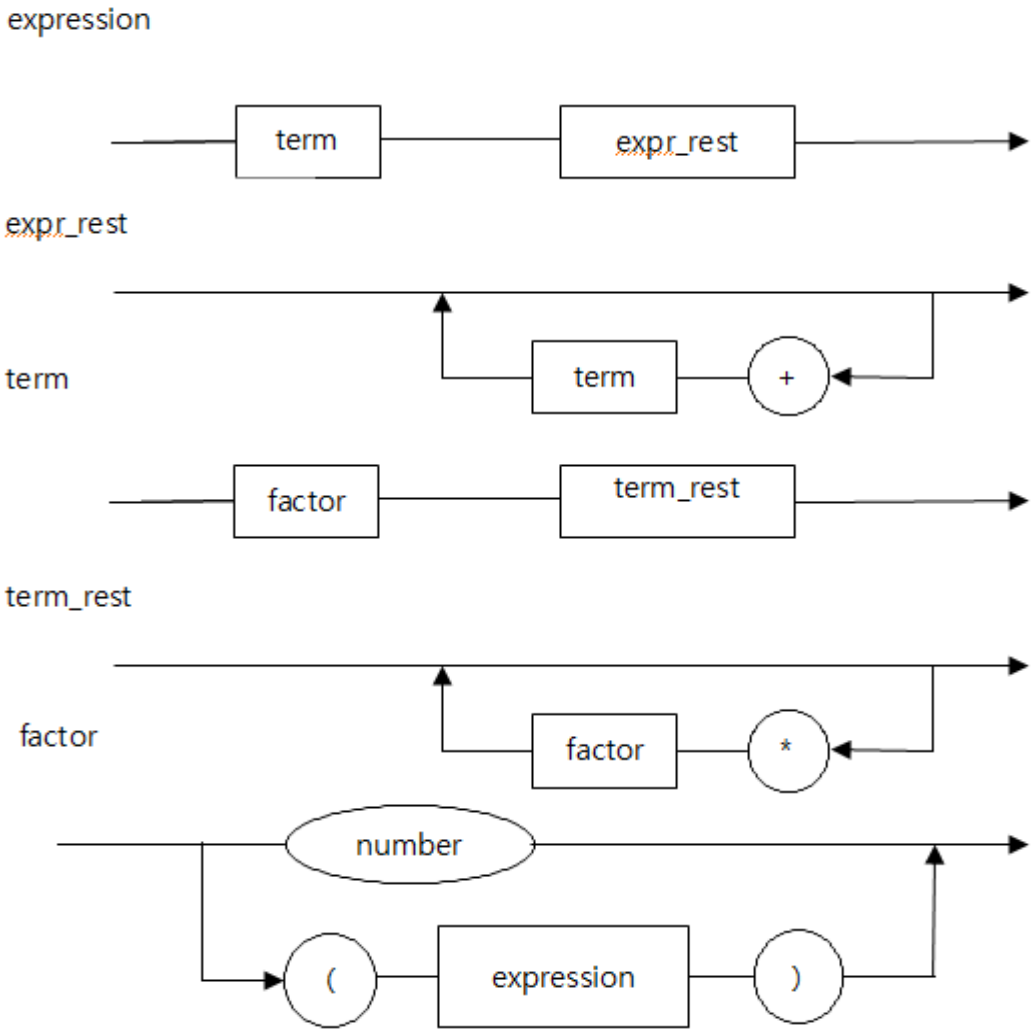
derivations

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

syntax tree

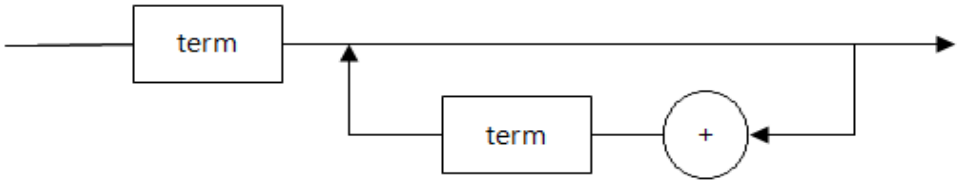


Syntax Graph

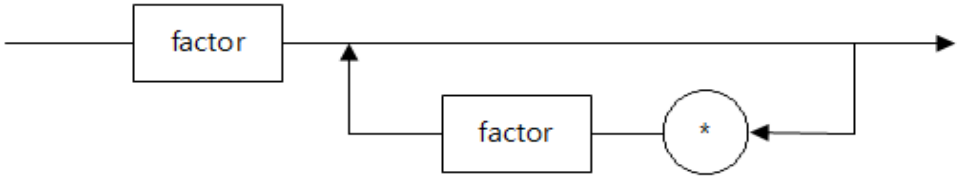


Syntax Graph for Simple Expressions

expression



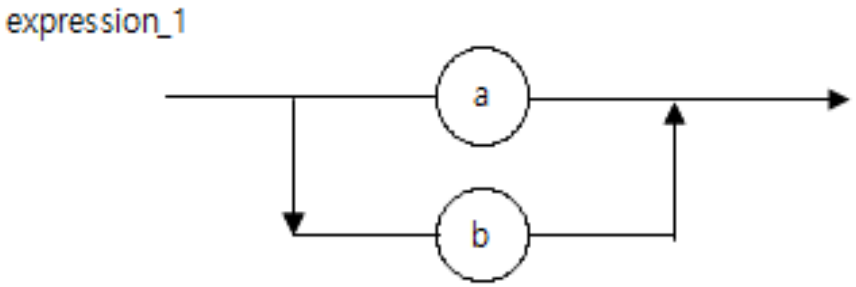
term



factor

Syntax Graph and Recursive-Descent Parser

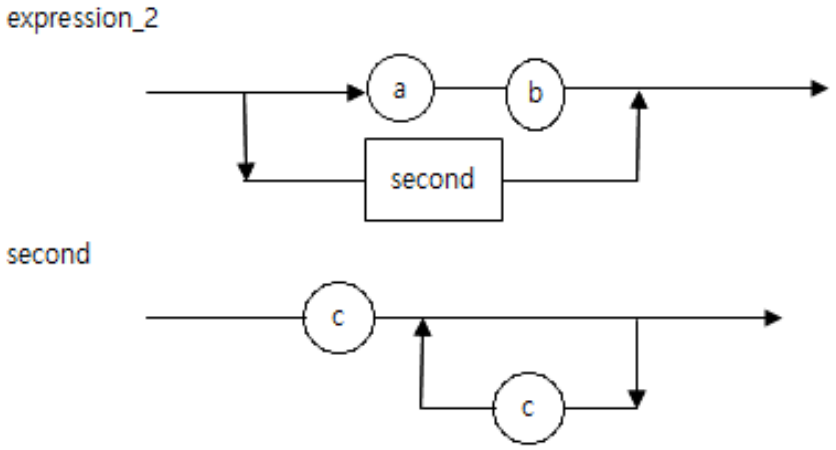
- { “a”, “b”}




```
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();
}
void error() {
    ……
}
```

- “ab” 나 “c…c” 인식

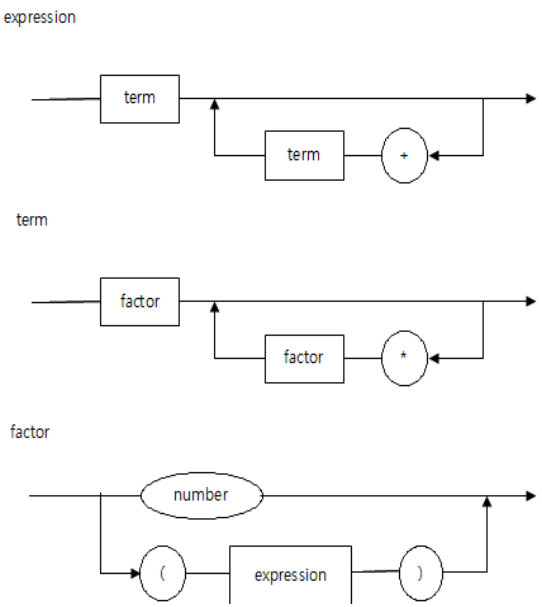



```
char ch=' ';
char get_token( ) {
    while (ch==' ')
            ch=getchar( );
    return (ch); }
void expression_2( ) {
    if (ch=='a')
            ch=get_token( );
            if (ch=='b')
                ch=get_token( );
            else
                error( );
           else
           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

간단한 수식을 위한 파싱 프로그램

```
enum {NULL, PLUS, STAR, NUMBER,
      LPAREN, RPAREN, END} token;
void get_token ( ) {
      // token = next token of input
}
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

입력문자	토큰종류 (token)
'0'~'9'	NUMBER
'+'	PLUS
'*'	STAR
'('	LPAREN
')'	RPAREL
EOF	END
기타	NULL

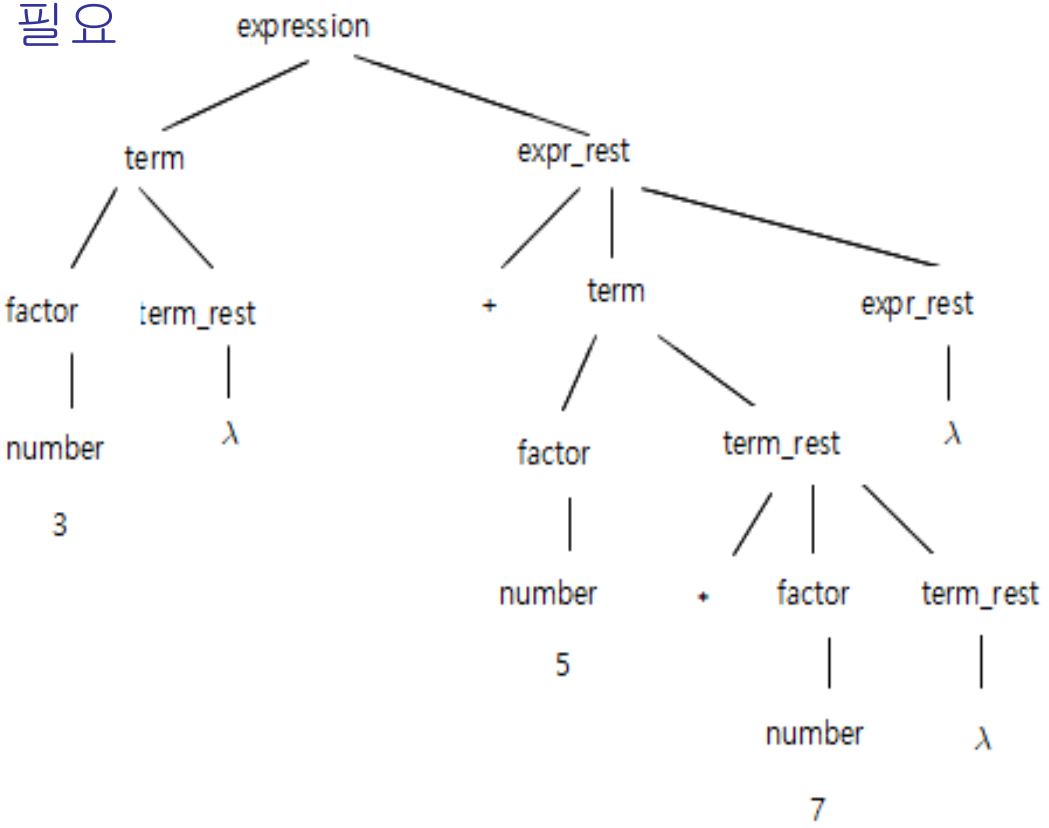
integer number : [0-9] [0-9]*

identifier: letter (letter | digit)*

Recursive-descent Parser 의 작동

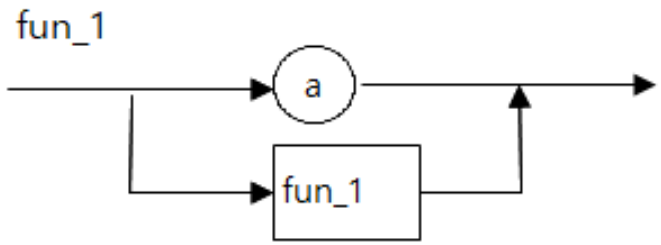
- 시작기호로 부터 하향식으로 호출

  main() --> expression() --> term() --> factor() --> expression() …
- 좌측 먼저 처리
- 한개의 토큰씩만 필요



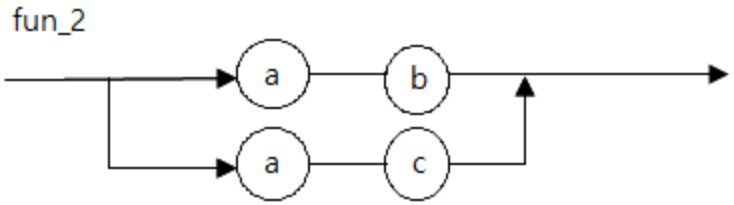
신택스 그래프의 제한사항

<경우-1>

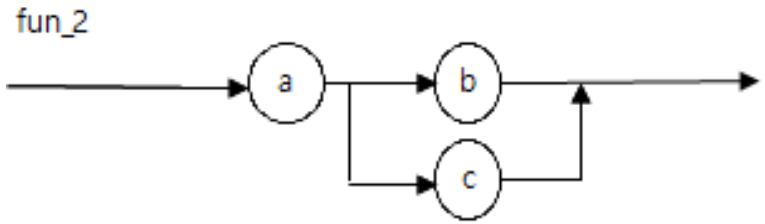



```
void fun_1 ( ) {
    if (token=='a')
        get_token( );
    else
        fun_1( );
}
```

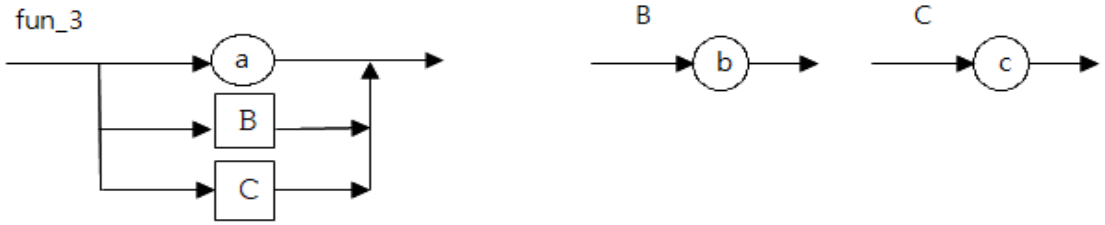
<경우-2> "ab" 나 "ac" 인식




```
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( );
}
```

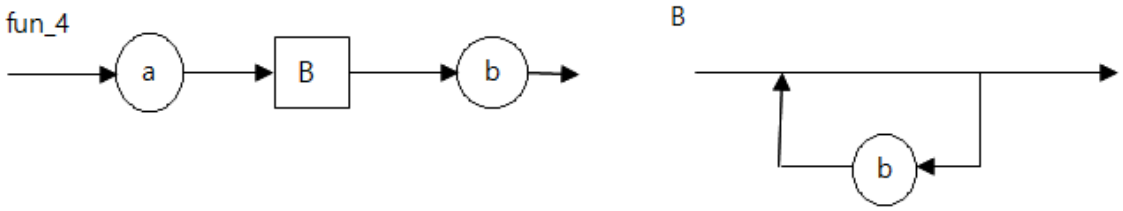


<경우-3> "a" 나 "b" 나 "c" 인식




```
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
}
```

<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( );
}
```

수식의 값 계산

```
int num;
enum NULL,NUMBER,PLUS,
    STAR,LP,RP,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);
}
```

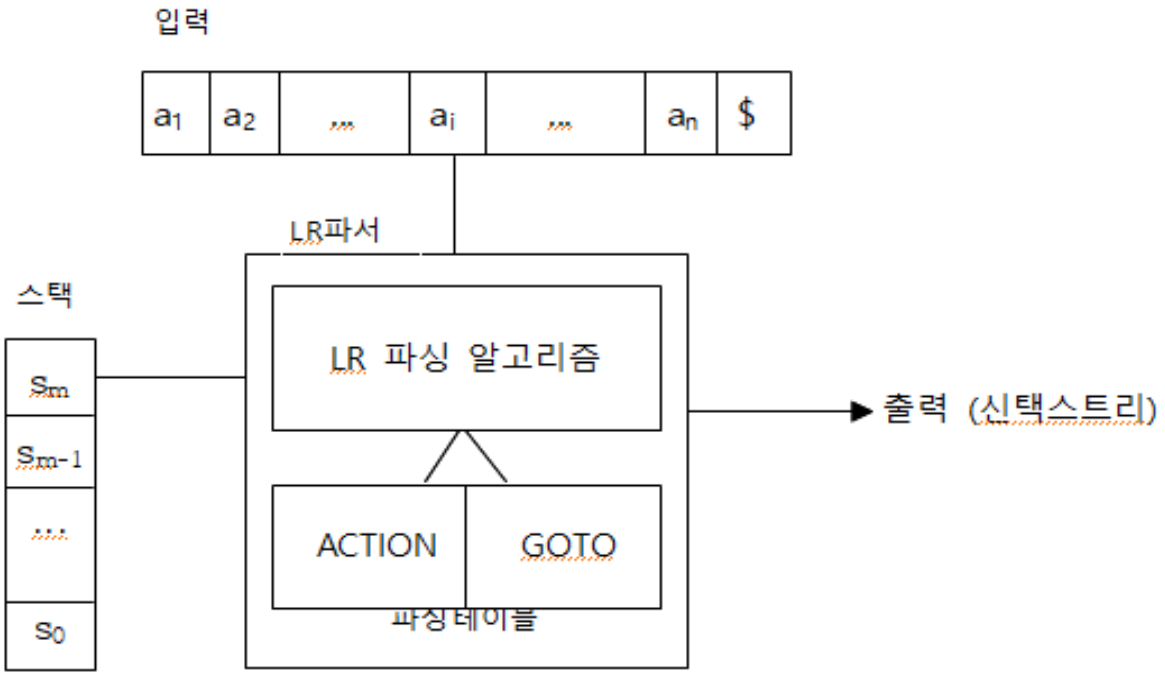
```
int factor ( ) {
    int result;
    if (token==NUMBER) {
        result=num
        get_token(); }
    else if (token==LP) {
         get_token();
         result=expression();
         if (token==RP)
                get_token();
         else
                error(2); }
    else
        error(1);
    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);
}
```

수식의 확장

- 연산자: 곱하기(*), 나누기(/)
- 피연산자: 정수 및 실수
- 결과값 인쇄:
  - 정수형 수식은 정수값, 실수형 수식은 실수값
- Warning messages
  - 혼합연산
- Error messages
- 함수나 token의 타입

```
typedef  enum {INT,FLT} kind;
struct { kind t;
         union { int i; float f;} val; }
```

LR Parser



상태번호

스택 : S_0 ... S_m

end-marker '$'

파싱테이블: ACTION[], GOTO[]

LR Parsing

파싱 동작:

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

파서상황 (configuration) Q : ($S_0 X_1 S_1 \ldots X_m S_m$, $a_i \ldots a_n\$$)

초기 파서상황 Q_0 : (0, $a_0 \ldots a_n$ $)

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

- "shift j":

  $(S_0X_1S_1\ldots X_mS_m, a_ia_{i+1}\ldots a_n\$) \rightarrow (S_0X_1S_1 \ldots X_mS_m\ a_i\ j,\ a_{i+1}\ldots a_n\$)$

- "reduce j":

  문법의 j 번째 규칙: 'A ::= α', r=|α|,

  $(S_0\ X_1S_1 \ldots X_mS_m,\ a_i\ldots a_n\$) \rightarrow$

  $(S_0\ X_1S_1 \ldots X_{m-r}S_{m-r},\ a_i\ldots a_n\$) \rightarrow$

  $(S_0\ X_1S_1 \ldots X_{m-r}S_{m-r}\ Ak,\ a_i\ldots a_n\$)$, 여기서 GOTO[s_{m-r}][A]=k

- "accept" :성공적 종료
- "error" : 에러 처리

LR Parsing Algorithm

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

LR문법과 LR Parsing Tables

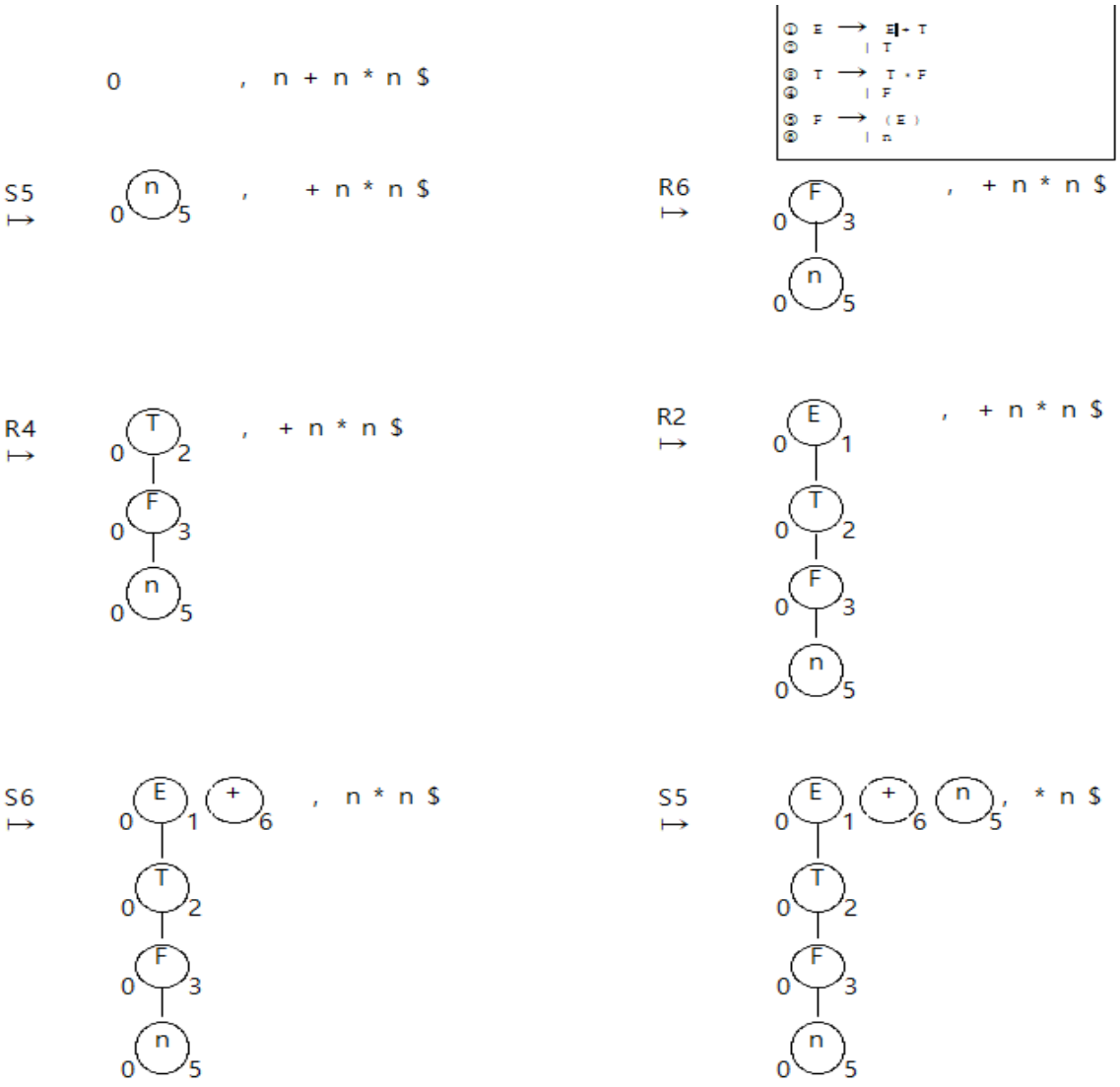
① E → E + T
② | T
③ T → T * F
④ | F
⑤ F → (E)
⑥ | n

테이블 / 상태번호	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			

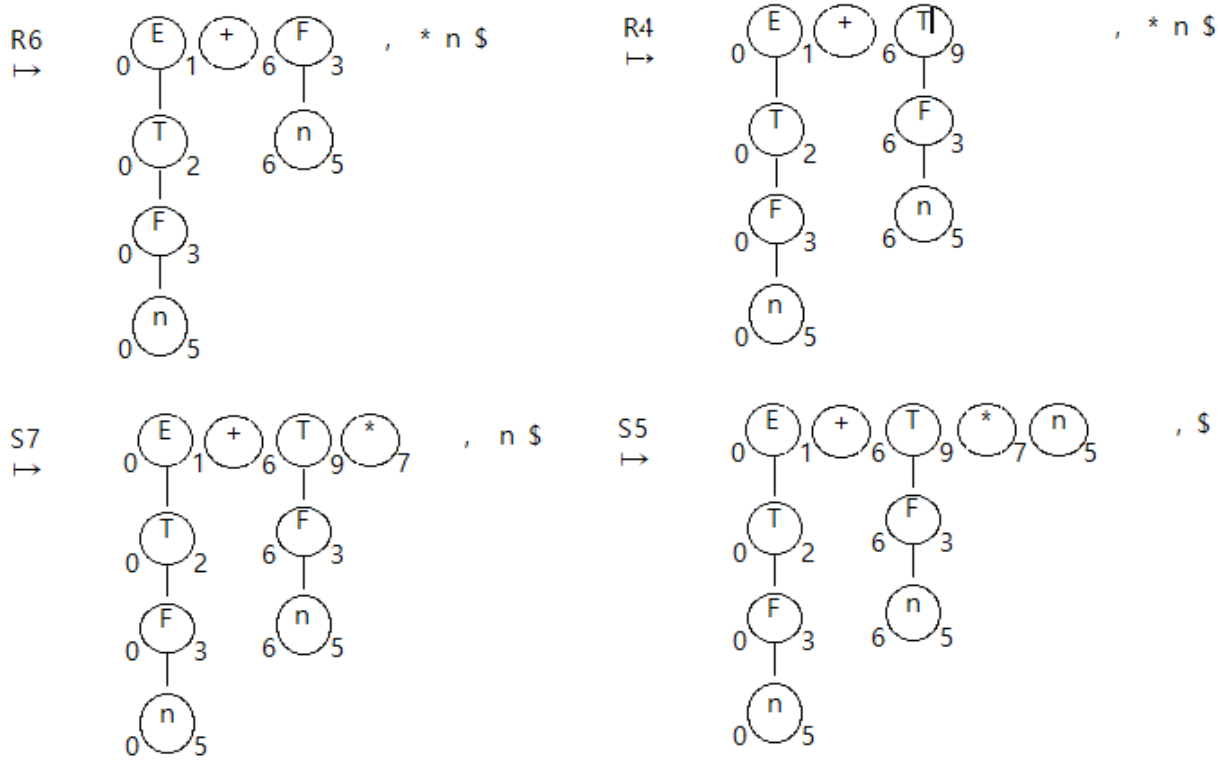
n+n*n 의 파싱과정

	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		

Parse Tree 생성 개념(1)



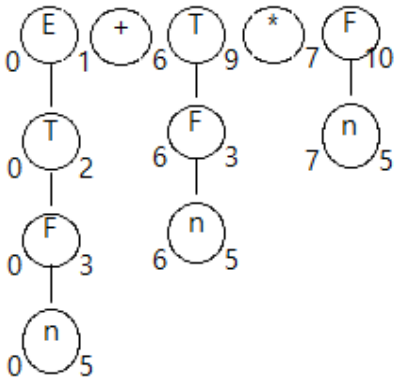
Parse Tree 생성 개념 (2)



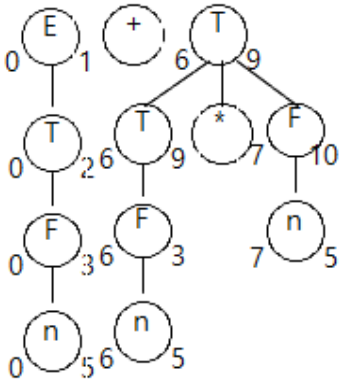
① E → E + T
② | T
③ T → T * F
④ | F
⑤ F → (E)
⑥ | n

Parse Tree 생성 개념 (3)

R6 ↦

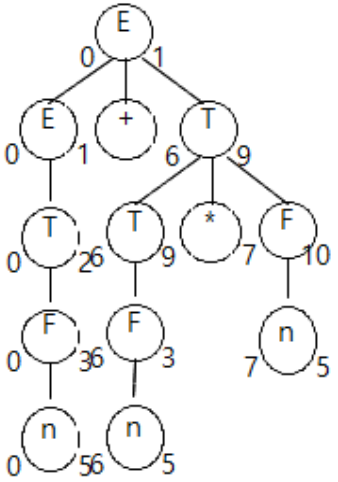


, $ R3 ↦



, $

R1 ↦



, $ acc ↦

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

LR Parser 특징

- SLR, LALR, 및 LR 파서 : 파싱 알고리즘은 동일

- LR 파싱 테이블
  문법으로부터 제작
  파싱 테이블 크기: '12줄*심볼갯수'
  C 언어: 500*500 정도 이상

LR Parser Program (1)

```
#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 main () {
        yyparse();
}
```

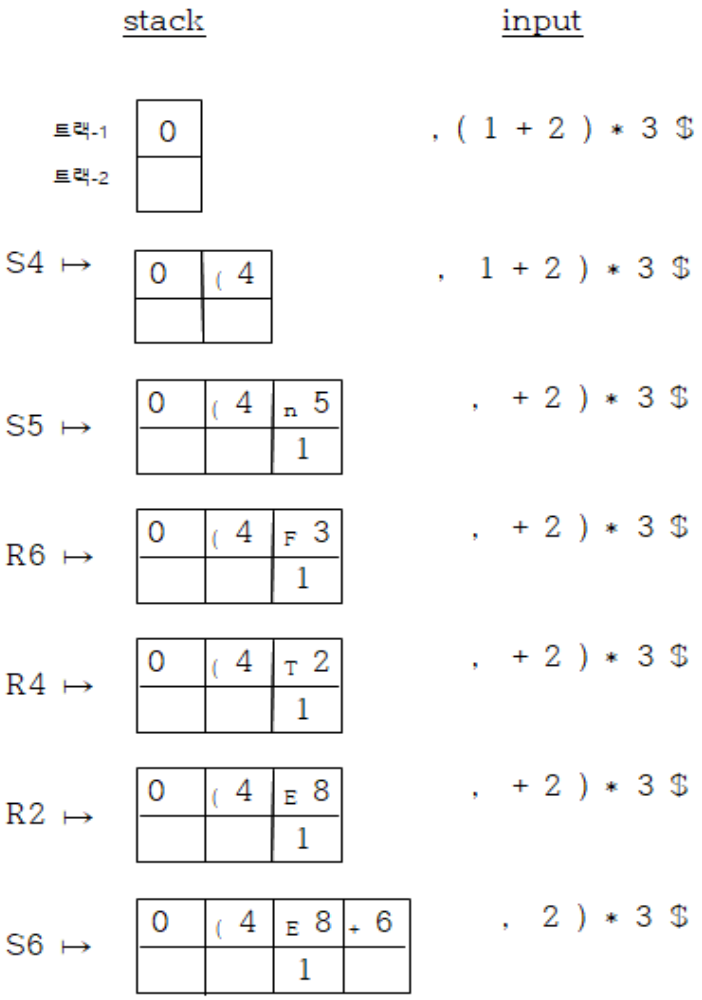
LR Parser Program (2)

```
int yyparse() {
        int i;
        stack[++top]=0;                          // initial state
        sym=yylex();
        do {    i=action[stack[top]][sym-256];   // get relation
                if (i==ACC)
                        printf("success !₩n");
                else if (i>0) shift(i);          // shift
                else if (i<0) reduce(-i);        // reduce
                else yyerror(); }
        while (i!=ACC);
}
void push(int l) {
        stack[++top]=i; }
void shift(int l) {
        push(i); sym=yylex(); }
void reduce(int l) {
        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); }
```

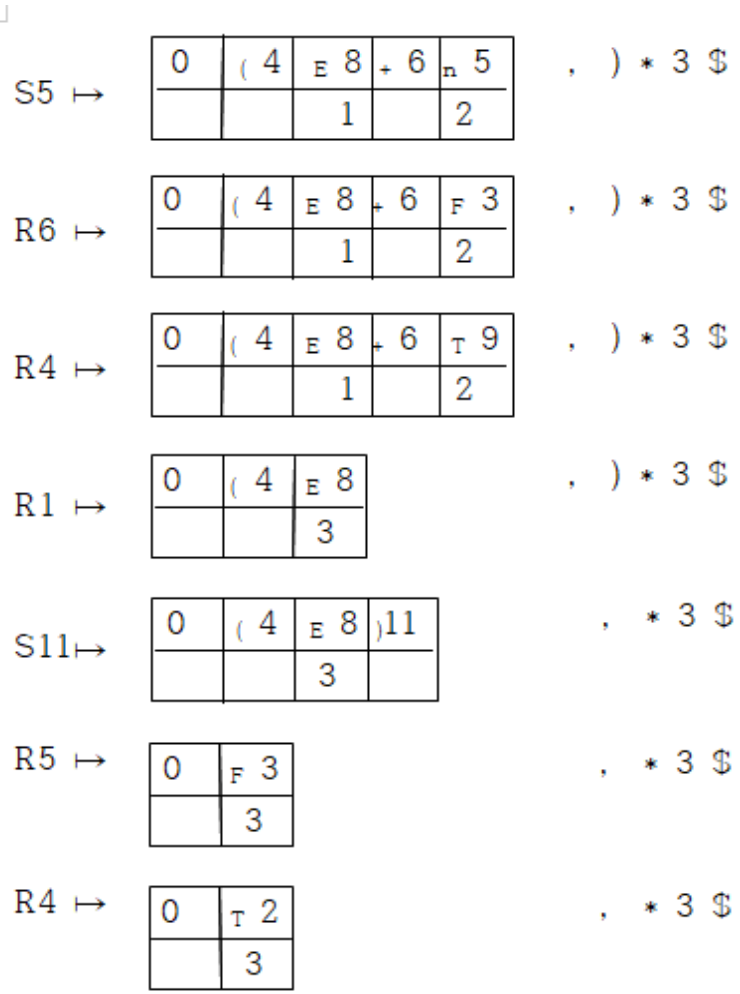
LR Parser Program (3)

```
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();
}
```

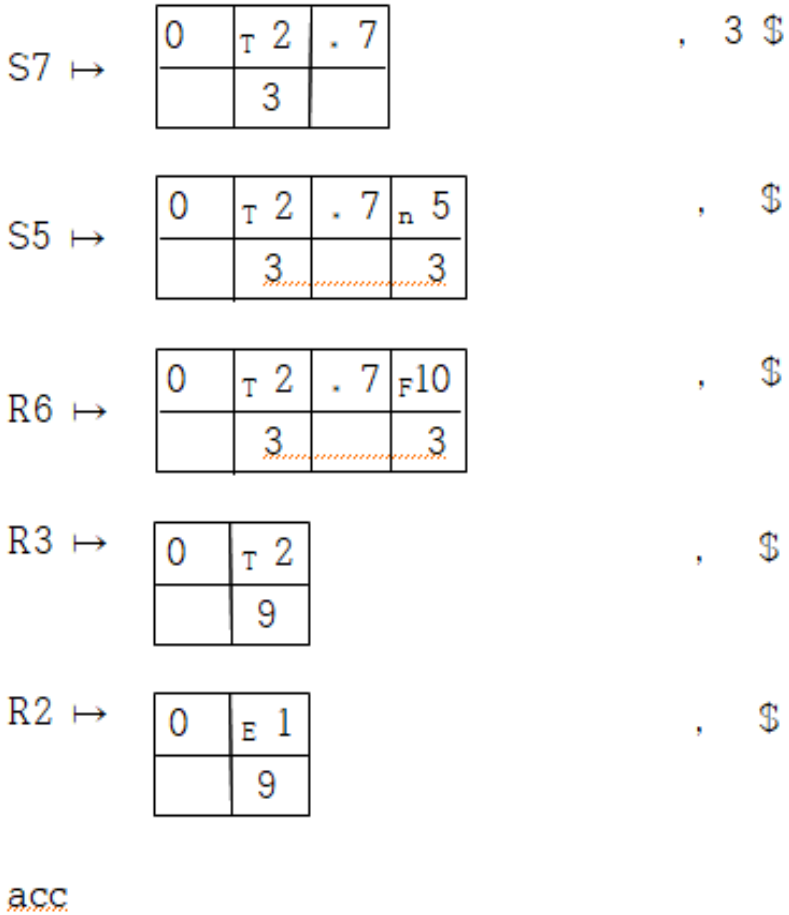
LR Parser 에서 수식의 값계산 방법(1)



LR Parser 에서 수식의 값계산 방법(2)



LR Parser 에서 수식의 값계산 방법(3)



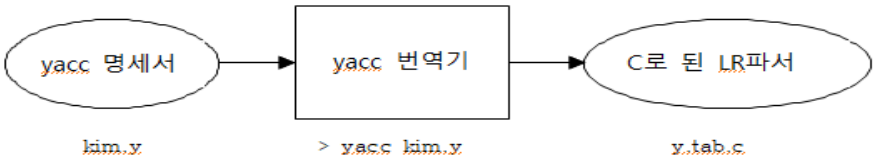
C Program(1)

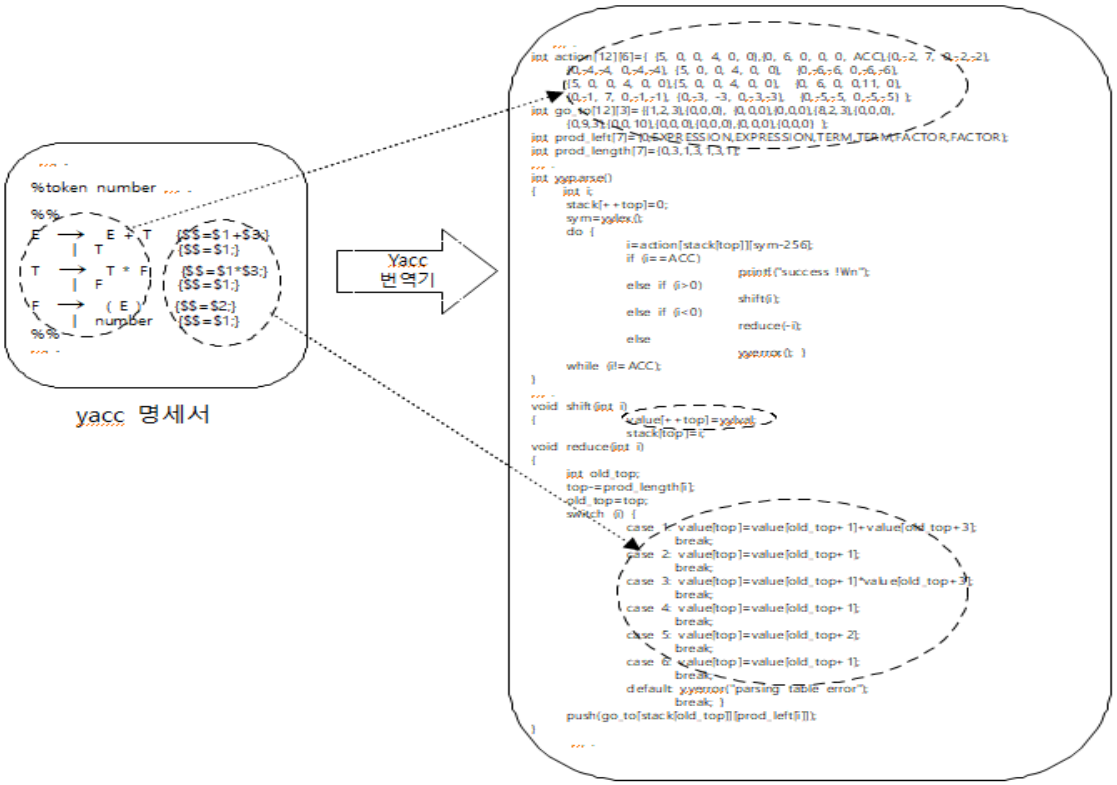
```
char yytext[32];
int yylval;
// 생략 ...
void shift(int l) {
        push(i);
        value[top]=yylval;
        sym=yylex(); }
void reduce(int l) {
        // 생략
        switch (i) {   // 규칙번호에 따른 수식 값 계산
                case 1: value[top]=value[old_top+1]+value[old_top+3];
                        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;
        } }
```

C Program(2)

```
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 ...
}
```

Yacc 번역기


yacc 명세서
yacc 번역기
C로 된 LR파서
kim.y
> yacc kim.y
y.tab.c


%token number
%%
E → E + T | T
T → T * F | F
F → (E) | number
%%
{$$=$1+$3;}
{$$=$1;}
{$$=$1*$3;}
{$$=$1;}
{$$=$2;}
{$$=$1;}
yacc 명세서
Yacc 번역기
int action[12][6]={ {5, 0, 0, 4, 0, 0},{0, 6, 0, 0, 0, ACC},{0,-2, 7, 0,-2,-2},
int go_to[12][3]= {{1,2,3},{0,0,0}, {0,0,0},{0,0,0},{8,2,3},{0,0,0},
int prod_left[7]={0,E,E,T,T,F,F};
int prod_length[7]={0,3,1,3, 1,3, 1};
int yyparse()
stack[++top]=0;
sym=yylex();
i=action[stack[top]][sym-256];
if (i==ACC)
printf("success !Wn");
else if (i>0)
shift(i);
else if (i<0)
reduce(-i);
else
yyerror(); }
while (i!= ACC);
void shift(int i)
value[++top]=yylval;
stack[top]=i;
void reduce(int i)
int old_top;
top-=prod_length[i];
old_top=top;
switch (i) {
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; }
push(go_to[stack[old_top]][prod_left[i]]);

Yacc 프로그램

- Yacc 명세서

  선언부 (delcarations)

  %%

  문법과 번역규칙부 (rules)

  %%

  보조 프로그램부 (supporting programs)

Yacc 작성 (1)

```
%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();
}
```

문법 G_2

$$
\begin{aligned}
E &\rightarrow E + T \\
&\mid T \\
T &\rightarrow T * F \\
&\mid F \\
F &\rightarrow (E) \\
&\mid 0 \\
&\mid \ldots \\
&\mid 9
\end{aligned}
$$

Yacc 작성 (2)

```
%{
#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
```

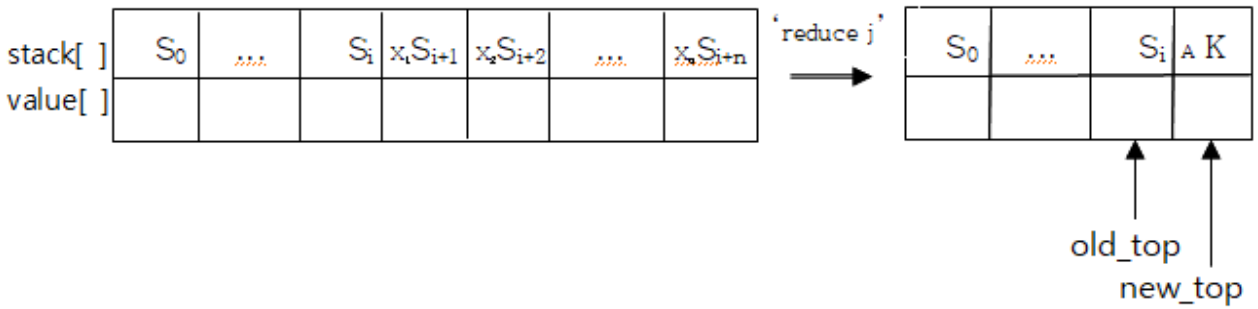
action rules

Yacc 사용방법

번역규칙: A ::= α1 | α2 | ... |αn

```
A    : α1   {번역규칙1}
     | α2   {번역규칙2}
     | ...      ...
     | αn   {번역규칙n}
```

'reduce' 동작



$$, $1, $2, $3 ... 등의 의미