

Reference: Tom Niemann, LEX & YACC. <http://epaperpress.com/lexandyacc>

목차

Flex

- 어휘분석기 생성기
- 정규표현식
- Flex 입력 파일 작성
- Flex 사용 예

Bison

- 파서 생성기
- 수식문법과 파서 생성
- yylval, \$\$, \$1, \$2, ...
- Union 타입 yylval

T 언어

- 파서 생성
- 추상구문트리 생성
- 해석기
- TAC(three address code) 생성
- 컴파일러(x86)

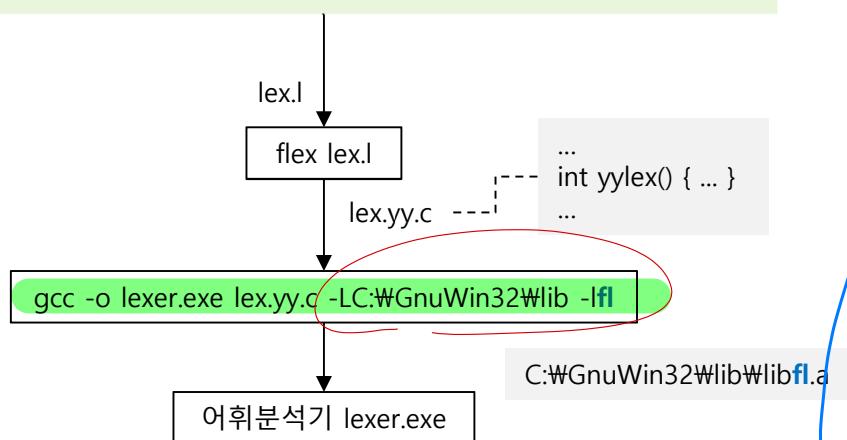
Lex #0: flex regular expression

Flex 패턴	설명
[0123456789]	[와] 내부에 나열된 문자들(0, 1, 2, 3, 4, 5, 6, 7, 8, 9) 중 임의의 한 개 문자
[0-9]	0부터 9까지 문자들(0, 1, 2, 3, 4, 5, 6, 7, 8, 9) 중 임의의 한 개 문자. [와] 내에서 -는 range의 의미로 해석
[0-9] +	0, 1, 2, 3, 4, 5, 6, 7, 8, 9의 문자들 중 임의의 한 개 문자가 1회 이상 출현하는 문자열
[a-z]	a부터 z까지 문자들(a, b, c, ..., z) 중 임의의 한 개 문자
[^0-9]	0, 1, 2, 3, 4, 5, 6, 7, 8, 9의 문자들 이외의 임의의 한 개 문자 의미. [와] 내 첫 위치 출현한 ^는 negation으로 해석
[a-zA-Z]	a, b, c, ..., z, 0, 1, 2, ..., 9의 문자들 중 임의의 한 개 문자
[a-zA-Z]*	a, b, c, ..., z, 0, 1, 2, ..., 9의 문자들 중 임의의 한 개 문자가 0회 이상 출현하는 문자열
[a-zA-Z][a-zA-Z]*	영어 소문자 하나로 시작하고, 그 뒤에 소문자 및 숫자 문자들 중 임의의 문자가 0회 이상 출현하는 문자열
.	₩n을 제외한 임의의 한 개 문자
₩.	문자 그대로의 .을 의미함. 역슬래쉬 기호 ₩이 없다면 ₩n을 제외한 임의의 한 개 문자를 의미
[-+*/(){}=:]	- , +, *, /, (), { }, =, ;의 문자들 중 임의의 한 개 문자를 의미
[₩t₩r₩n]	공백, 탭, carriage return, newline 문자들 중 임의의 한 개 문자
"int"	"와 "로 감싸인 문자열 int를 의미. "[0-9]+"와 "로 감싸인 문자열 그대로의 [0-9]+를 의미
"₩"G."₩" Hong"	문자열 그대로의 "G." Hong을 의미
([a-zA-Z]+) ([0-9]+)	[a-zA-Z]+ 혹은 [0-9]+를 의미
[0-9]+₩.	숫자 문자열이 출현하고 그 뒤에 문자 그대로의 .이 출현하는 문자열
[0-9]+(₩.)?	숫자 문자열이 출현하고 그 뒤에 문자 그대로의 .이 0회 출현하거나 1회 출현하는 문자열

Flex #1

lex.l (스캐너 기술 파일)

Pattern(정규표현식)	Action(C 코드)
%{	
#include <stdio.h>	
%}	
%%	
[0-9]+	{ printf("<NUM, %s>\n", yytext); }
[a-zA-Z][a-zA-Z0-9]*	{ printf("<ID, %s>\n", yytext); }
[]	{ }
\t	{ }
\n	{ }
.	{ printf("<%c, %c>\n", yytext[0], yytext[0]); }
%%	
int main(){	
yylex();	
return 0;	
}	



- Flex는 스캐너 기술 파일(예: lex.l)을 입력으로 받아 lex.yy.c를 생성하며, lex.yy.c 내에는 yylex() 함수가 정의되어 있음
 - Flex 입력 파일(스캐너 기술 파일) 작성 형식 → 저이브

정의부
%%
규칙부
%%
User code

- 한 규칙(rule)은 pattern과 action의 쌍으로 기술되며, pattern은 정규표현식(regular expression)으로, action은 C 코드로 작성됨
 - User code는 optional이며 lex.yy.c에 그대로 복사됨
 - 정의부(혹은 규칙부)에 %{와 }% 내 텍스트는 lex.yy.c로 복사됨
 - 생성된 스캐너는 정규표현식에 매치되는 문자열 출현 시 대응하는 C 코드를 실행함. 정규표현식에 매치된 문자열은 기정의된 character pointer 변수 yytext로 접근 가능

스캐너 생성 및 실행 예

```
flex lex.l  
gcc -o lexer.exe lex.yy.c -LC:\GnuWin32\lib -lfl  
lexer < input.txt
```

```
input.txt
```

```
int main(){  
    int score=93;  
}
```

```
<ID, int>
<ID, score>
<=, =>
<NUM, 93>
<; ;>
<}, }>
```

구문

Flex #2

lex.l (스캐너 기술 파일)

```
%{  
#include <stdio.h>  
%}  
  
%%  
"int"           { printf("<%s, INT>\n", yytext); }  
"if"            { printf("<%s, IF>\n", yytext); }  
"else"          { printf("<%s, ELSE>\n", yytext); }  
"("             { printf("<%s, LPAREN>\n", yytext); }  
")"             { printf("<%s, RPAREN>\n", yytext); }  
 "{"             { printf("<%s, LBRACE>\n", yytext); }  
 "}"             { printf("<%s, RBRACE>\n", yytext); }  
 ";"             { printf("<%s, SEMICOLON>\n", yytext); }  
 "="             { printf("<%s, ASSIGN>\n", yytext); }  
 ">="            { printf("<%s, GE>\n", yytext); }  
 "+"             { printf("<%s, ADD>\n", yytext); }  
 "*"             { printf("<%s, MUL>\n", yytext); }  
 [ \t\r\n]         { }  
 [0-9]+          { printf("<%s, NUM>\n", yytext); }  
 [a-zA-Z][a-zA-Z0-9]* { printf("<%s, ID>\n", yytext); }  
 .               { printf("%s => illegal\n", yytext); }  
 %%  
  
int main(){  
    yylex();  
    return 0;  
}
```

yylex()

- 입력에서 읽은 토큰 반환
- 파일 끝(end-of-file) 도달 시 0 반환

스캐너 생성 및 실행 예

```
flex lex.l  
gcc -o lexer.exe lex.yy.c -LC:\GnuWin32\lib -lfl  
lexer < input.txt
```

```
<int, INT>  
<main, ID>  
<(, LPAREN>  
<), RPAREN>  
<{, LBRACE>  
<int, INT>  
<value, ID>  
<=, ASSIGN>  
<567, NUM>  
<;, SEMICOLON>  
<if, IF>  
<(, LPAREN>  
<value, ID>  
<>=, GE>  
<0, NUM>  
<), RPAREN>  
<value, ID>  
<=, ASSIGN>  
<value, ID>  
<+, ADD>  
<3, NUM>  
<;, SEMICOLON>  
<else, ELSE>  
<value, ID>  
<=, ASSIGN>  
<5, NUM>  
<*, MUL>  
<value, ID>  
<;, SEMICOLON>  
<}, RBRACE>
```

input.txt

```
int main(){  
    int value=567;  
    if(value>=0) value=value+3;  
    else value=5*value;  
}
```

Flex #3

lex.l (스캐너 기술 파일)

```
%{  
#include <stdio.h>  
enum TokenName {  
INT=1,IF,ELSE,LPAREN,RPAREN,LBRACE,RBRACE,SEMICOLON,ASSIGN,GE,ADD,MUL,NU  
M,ID };  
char *tokenNameStr[] =  
{ "", "INT", "IF", "ELSE", "LPAREN", "RPAREN", "LBRACE", "RBRACE", "SEMICOLON", "ASSIGN", "GE", "A  
DD", "MUL", "NUM", "ID" };  
%}  
  
%%  
"int"           { return INT; }  
"if"            { return IF; }  
"else"          { return ELSE; }  
"("             { return LPAREN; }  
")"             { return RPAREN; }  
"{"             { return LBRACE; }  
"}"             { return RBRACE; }  
";"             { return SEMICOLON; }  
"="             { return ASSIGN; }  
">="            { return GE; }  
"+"             { return ADD; }  
"*"             { return MUL; }  
[ \t\r\n]          { }  
[0-9]+          { return NUM; }  
[a-zA-Z][a-zA-Z0-9]* { return ID; }  
.              { printf("'%s => illegal\n", yytext); }  
%  
  
int main(){  
    int tokenName;  
    while(tokenName=yylex()){ printf(<%s, %s>\n", yytext, tokenNameStr[tokenName]); }  
    return 0;  
}
```

TOKEN 이름만 정의

```
#define INT    1  
#define IF     2  
#define ELSE   3  
...  
#define ID    14
```

TOKEN을 읽고 Return 하는 코드

제작 대상 언어: C/C++
제작 도구: 편집기: Notepad++

스캐너 생성 및 실행 예

```
flex lex.l  
gcc -o lexer.exe lex.yy.c -LC:\GNuWin32\lib -lfl  
lexer < input.txt
```

```
<int, INT>  
<main, ID>  
<(, LPAREN>  
<), RPAREN>  
<{, LBRACE>  
<int, INT>  
<value, ID>  
<=, ASSIGN>  
<567, NUM>  
<;, SEMICOLON>  
<if, IF>  
<(, LPAREN>  
<value, ID>  
<>, GE>  
<0, NUM>  
<), RPAREN>  
<value, ID>  
<=, ASSIGN>  
<value, ID>  
<+, ADD>  
<3, NUM>  
<;, SEMICOLON>  
<else, ELSE>  
<value, ID>  
<=, ASSIGN>  
<5, NUM>  
<*, MUL>  
<value, ID>  
<;, SEMICOLON>  
<}, RBRACE>
```

```
input.txt  
  
int main(){  
    int value=567;  
    if(value>=0) value=value+3;  
    else value=5*value;  
}
```

yylex()

- 입력에서 읽은 토큰 반환
- 파일 끝(end-of-file) 도달 시 0 반환

Flex #4

lex.l (스캐너 기술 파일)

```
%{  
#include <stdio.h>  
#include <string.h>  
enum TokenName { INT=258, IF, ELSE, GE, NUM, ID };  
char *tokenNameStr[] = { "INT", "IF", "ELSE", "GE", "NUM", "ID" };  
%}  
  
%%  
"int"           { return INT; }  
"if"            { return IF; }  
"else"          { return ELSE; }  
[0{}:=+*]        { return yytext[0]; }  
">="             { return GE; }  
[ \t\r\n]+         { ; }  
[0-9]+           { return NUM; }  
[a-zA-Z][a-zA-Z0-9]* { return ID; }  
.               { printf("%s => illegal", yytext); }  
%%  
  
int main(){  
    int tokenName;  
    while(tokenName=yylex()){  
        if(strstr("0{}:=+*", yytext)) printf("<%c, %c>\n", tokenName, tokenName);  
        else printf("<%s, %s>\n", tokenNameStr[tokenName-258], yytext);  
    }  
    return 0;  
}
```

스캐너 생성 및 실행 예

```
flex lex.l  
gcc -o lexer.exe lex.yy.c -LC:\GnuWin32\lib -lfl  
lexer < input.txt
```

```
<INT, int>  
<ID, main>  
<(>  
<), )>  
<{}, {}>  
<INT, int>  
<ID, value>  
<=, =>  
<NUM, 567>  
<; ;>  
<IF, if>  
<(>  
<ID, value>  
<GE, >=>  
<NUM, 0>  
<), )>  
<ID, value>  
<=, =>  
<ID, value>  
<+, +>  
<NUM, 3>  
<; ;>  
<ELSE, else>  
<ID, value>  
<=, =>  
<NUM, 5>  
<*, *>  
<ID, value>  
<; ;>  
<}, }>
```

input.txt

```
int main(){  
    int value=567;  
    if(value>=0) value=value+3;  
    else value=5*value;  
}
```

Bison

Declaration section

- 모든 토큰 타입 명칭 선언 필요 (단일 문자 리터럴 토큰은 제외)
%token symbol1 symbol2 ...
- Non-terminal 심볼도, 그 semantic value의 타입 지정 필요시, 선언 필요
- 첫 규칙의 LHS가 시작 심볼 아닌 경우, 시작 심볼 선언 필요
%start symbol
- Semantic value가 여러 타입이면 %union 선언 사용 및
%token <type> symbol1 symbol2 ...
%type <type> nonterminalsymbol1 nonterminalsymbol2 ...
- %type → nonterminal symbol의 semantic value의 타입 명시
- %start → 시작 심볼 명시
- %noassoc → nonassociative인 terminal symbol 명시
- %left → left-associative인 terminal symbol 명시
- %right → right-associative인 terminal symbol 명시
- %token → precedence나 associative에 무관한 terminal symbol 명시
- %union → semantic value가 취할 수 있는 여러 타입들 명시
- int yyparse(void); → 파싱 성공 시 0 반환, 실패 시 1 반환
- int yylex(void); → 입력 끝이면 0 반환, 토큰 발견 시 토큰 타입에 해당하는 양수 반환
- 토큰의 semantic value는 yylval에 저장 필요
- 여러 타입을 갖는 semantic value 사용 시 yylval의 타입은 %union 선언으로 생성되는 union임
- \$\$ → 현재 rule의 LHS 심볼에 대한 semantic value
- \$n → 현재 rule의 RHS의 n번째 심볼에 대한 semantic value(예: \$1, \$2, ...)
- Shift/reduce conflict 시 디폴트로 shift 선택 (예: dangling-else)
- Reduce/reduce conflict의 경우 순서 상 먼저 기술된 규칙 선택됨 → 그러나 위험하므로 이런 충돌은 제거 필요

Parser generator

표 고체에 선택의 문법을 정의하고
파서로 만들어주세요.

문법 $G = (V, T, S, P)$

$V = \{S, E, T, F, D\}$,

$T = \{;, +, *, (,), 0, 1, 2, 3, 4, 5, 6, 7, 8, 9\}$

연습

- 문법 G가 생성하는 문장의 예를 제시하시오

$S \rightarrow E ;$
 $E \rightarrow E + T$
 $E \rightarrow T$
 $T \rightarrow T * F$
 $T \rightarrow F$
 $F \rightarrow (E)$
 $F \rightarrow D$
 $D \rightarrow 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9$

yacc.y → **bison** → yacc.tab.c

```
int yylex() {  
    ...  
}
```

파서 생성기
(예: yacc, **bison**)

```
...  
int yyparse() {  
    ... yylex()  
}  
int main() {  
    ... yyparse()  
}...
```

$1 + 2 * ((3 + 4) * 5 + 6) + 7;$ → 문법 G 에 대한 parser → parsing 성공/실패

#1: 수식 문법과 파서 생성

yacc.y ← 문법 **G**의 grammar 파일

```
%{
#include <stdio.h>
int yylex();
int yyerror(const char*);
%}

%token D 터미널 토큰이다. (마지막 토큰)

%%

S : E ';' 선언부
E : E '+' T ;
E : T ;
T : T '*' F ;
T : F ;
F : '(' E ')' ;
F : D ;

%%

int yylex(){
    int c=getchar();
    if(c>='0' && c<='9') return D; // Digit
    return c;
}

int main(){
    if(0==yparse()) printf("parsing: success");
    else printf("parsing: fail");
    return 0;
}
```

(Bison) grammar 파일 형식

```
%{ 프롤로그
%} 선언부
%% 문법규칙
%% 예필로그
```

- 단일 문자 terminal symbol 이외의 terminal symbol 명칭 선언(%token)
- Nonterminal symbol 명칭 선언(%type)
- 연산자 우선순위 선언(%left, %right, %nonassoc)
- 심볼의 semantic value의 데이터 타입 선언 등

이거 버튼은
할 줄 알아야 함

이걸로
버튼을

Bison에서의 문법 **G 표현**

```
S : E ';' ;
E : E '+' T ;
E : T ;
T : T '*' F ;
T : F ;
F : '(' E ')' ;
F : D ;

S : E '!';
E : E '+' T ;
E : T ;
T : T '*' F ;
T : F ;
F : '(' E ')' ;
F : D ;
```

해당 코드는 저에게 벤처 Grammar **G**

문법 **G 파서 생성 및 파싱 예**

bison yacc.y
gcc -o parser yacc.tab.c -LC:\GnuWin32\lib -ly
parser < input1.txt
parsing: success
parser < input2.txt
syntax error
parsing: fail
parser < input3.txt
syntax error
parsing: fail

input1.txt
 $1+2*(3+4)*5+6)+7;$

input2.txt
 $1+2-3;$

input3.txt
 $1+2;$

공백(space) 없어야 함

수식 용법에 맞음

문법에 고려하지
않아서 처리 실패

yyparse()

- 입력에서 읽은 토큰 반환
- 파일 끝(end-of-file) 도달 시 0 반환

yyerror()

- Parsing 성공 시 0을 반환
- Syntax error 발견 시 yyerror() 호출

yacc.y → bison → yacc.tab.c

```
...
int yyparse() {
    ... yylex() ...
}
int main() {
    ... yyparse()
}
...
```

#2: 수식 문법과 파서 생성

yacc.y ← grammar 파일

```
%{
#include <stdio.h>
int yylex();
int yyerror(const char*);
%}

%token D

%%
S : E ';' { printf("S -> E\n"); }
| ;
E : E '+' T { printf("E -> E + T\n"); }
| T { printf("E -> T\n"); }
| ;
T : T '*' F { printf("T -> T * F\n"); }
| F { printf("T -> F\n"); }
| ;
F : '(' E ')' { printf("F -> ( E )\n"); }
| D { printf("F -> D\n"); }
| ;
%%

int yylex(){
    int c=getchar();
    if(c>='0' && c<='9') return D; // Digit
    return c;
}
int main(){
    if(0==yparse()) printf("parsing: success");
    else printf("parsing: fail");
    return 0;
}
```

Action

- 해당 규칙이 적용될 때 실행되는 코드
- 대부분의 경우 규칙의 마지막에 작성됨
- 규칙의 중간에도 가능(mid-rule action)

입력값

input1.txt

1+2*3;

$S \Rightarrow E$
 $\Rightarrow E + T$
 $\Rightarrow T + T$
 $\Rightarrow F + T$
 $\Rightarrow D + T$
 $\Rightarrow I + T$
 $\Rightarrow I + T + F$
 $\Rightarrow 1 + 2 * 3$

파서 생성 및 파싱 예

bison yacc.y

gcc -o parser yacc.tab.c -LC:\GnuWin32\lib -ly
parser < input1.txt

F -> D
T -> F
E -> T
F -> D
T -> F
F -> D
T -> T * F
E -> E + T
S -> E

입력 문장 1+2*3;의 파싱 과정에서
적용된 규칙들을 순차 출력
(우단유도의 역순)

반대(후위)로 적용이 된다.

▼

parsing: success

$S \rightarrow E;$
 $E \rightarrow E + T$
 $E \rightarrow T$
 $T \rightarrow T * F$
 $T \rightarrow F$
 $F \rightarrow (E)$
 $F \rightarrow D$
 $D \rightarrow 0 | 1 | 2 | \dots | 9$

우단유도	적용된 규칙
S	$S \rightarrow E ;$
$\Rightarrow E;$	$E \rightarrow E + T$
$\Rightarrow E + T;$	$T \rightarrow T * F$
$\Rightarrow E + T * F;$	$F \rightarrow D$
$\Rightarrow E + T * D;$	$T \rightarrow F$
$\Rightarrow E + T * 3;$	$F \rightarrow D$
$\Rightarrow E + F * 3;$	$E \rightarrow T$
$\Rightarrow E + D * 3;$	$T \rightarrow F$
$\Rightarrow E + 2 * 3;$	$F \rightarrow D$
$\Rightarrow T + 2 * 3;$	$E \rightarrow T$
$\Rightarrow F + 2 * 3;$	$T \rightarrow F$
$\Rightarrow D + 2 * 3;$	$F \rightarrow D$
$\Rightarrow 1 + 2 * 3;$	

$1 + 2 * 3$

좌단 유도

$$\begin{aligned}
 S &\Rightarrow E; \\
 &\Rightarrow E + T; \\
 &\Rightarrow T + T; \\
 &\Rightarrow F + T; \\
 &\Rightarrow D + T; \\
 &\Rightarrow 1 + T; \\
 &\Rightarrow 1 + T * F; \\
 &\Rightarrow 1 + F * F; \\
 &\Rightarrow 1 + D * F; \\
 &\Rightarrow 1 + 2 * F; \\
 &\Rightarrow 1 + 2 * D; \\
 &\Rightarrow 1 + 2 * 3;
 \end{aligned}$$

Σ (D, 1)

$$\begin{aligned}
 S &\Rightarrow E; \\
 &\Rightarrow E + T; \\
 &\Rightarrow T + T; \\
 &\Rightarrow F + T; \\
 &\Rightarrow D + T; \\
 &\Rightarrow D + T * F; \\
 &\Rightarrow D + F * F; \\
 &\Rightarrow D + D * F; \\
 &\Rightarrow D + D * D;
 \end{aligned}$$

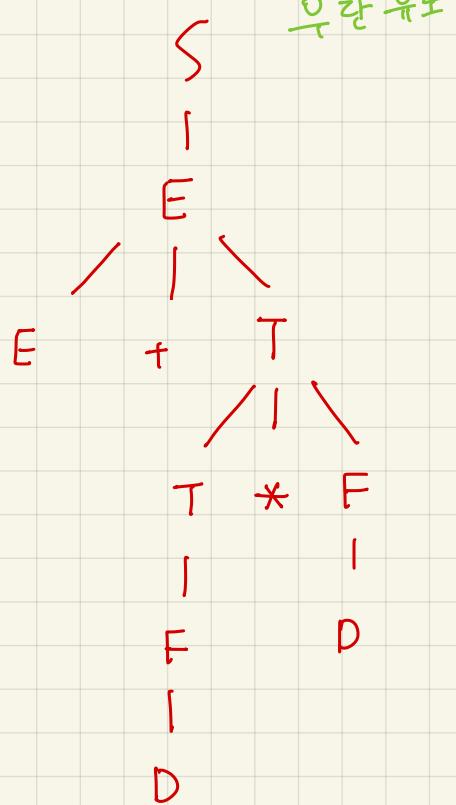
우단 유도

우단 유도

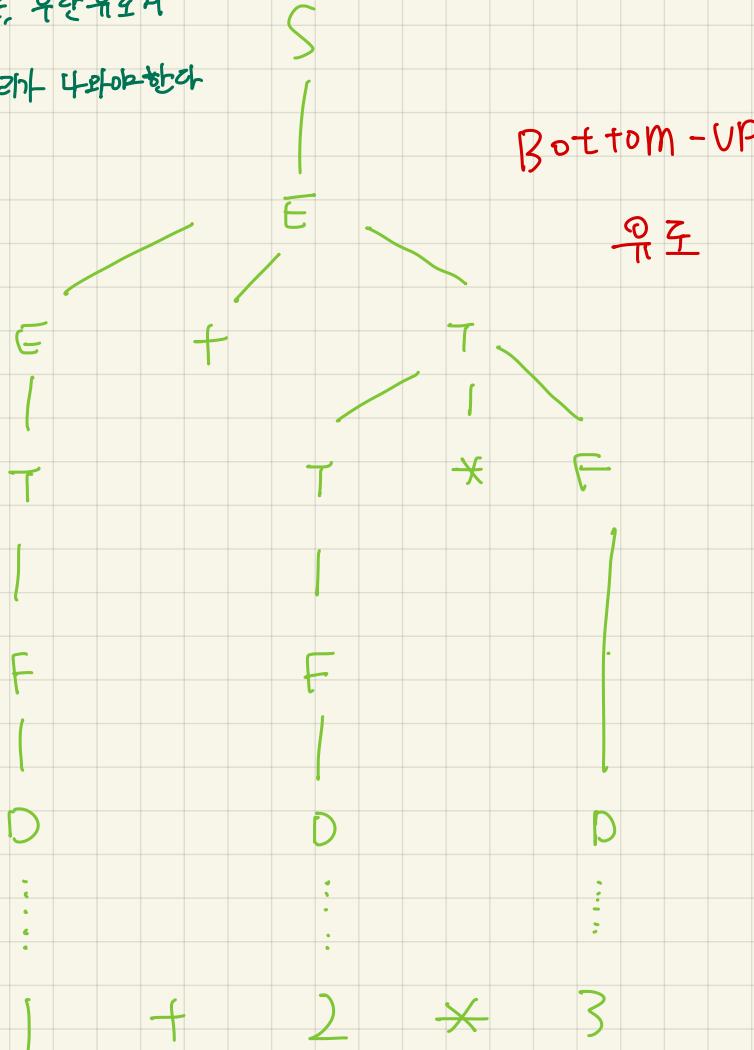
$$\begin{aligned}
 S &\Rightarrow E; \\
 &\Rightarrow E + T; \\
 &\Rightarrow E + (T * F); \\
 &\Rightarrow E + T * D; \\
 &\Rightarrow E + F * D; \\
 &\Rightarrow E + D * D; \\
 &\Rightarrow T + D * D; \\
 &\Rightarrow F + D * D; \\
 &\Rightarrow D + D * D;
 \end{aligned}$$

우단 유도

집에서부터
거기로 차면
Bottom-up 유도



좌측 유로, 우측 유로
같은 조건 나와야 한다



Lexer generator, parser generator

문법 $G = (V, T, S, P)$

$V = \{S, E, T, F, D\}$,
 $T = \{;, +, *, (,), 0, 1, 2, 3, 4, 5, 6, 7, 8, 9\}$

$S \rightarrow E ;$
 $E \rightarrow E + T$
 $E \rightarrow T$
 $T \rightarrow T * F$
 $T \rightarrow F$
 $F \rightarrow (E)$
 $F \rightarrow D$
 $D \rightarrow 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9$

여기서 토큰은
 $; + * ()$ 등이...
 각각에 일정한 값을

```
enum yytokentype {
    D = 258
};
typedef int YYSTYPE;
YYSTYPE yylval;
...
int yyparse() {
    ... yylex() ...
}
int main() {
    ... yyparse() ...
}
```

다음 토큰을
 가져오는 함수

문법 G

```
%token D
%%
S : E ';' ;
E : E '+' T ;
E : T ;
T : T '*' F ;
T : F ;
F : '(' E ')' ;
F : D ;
%%
int main() { ... yyparse() ... }
```

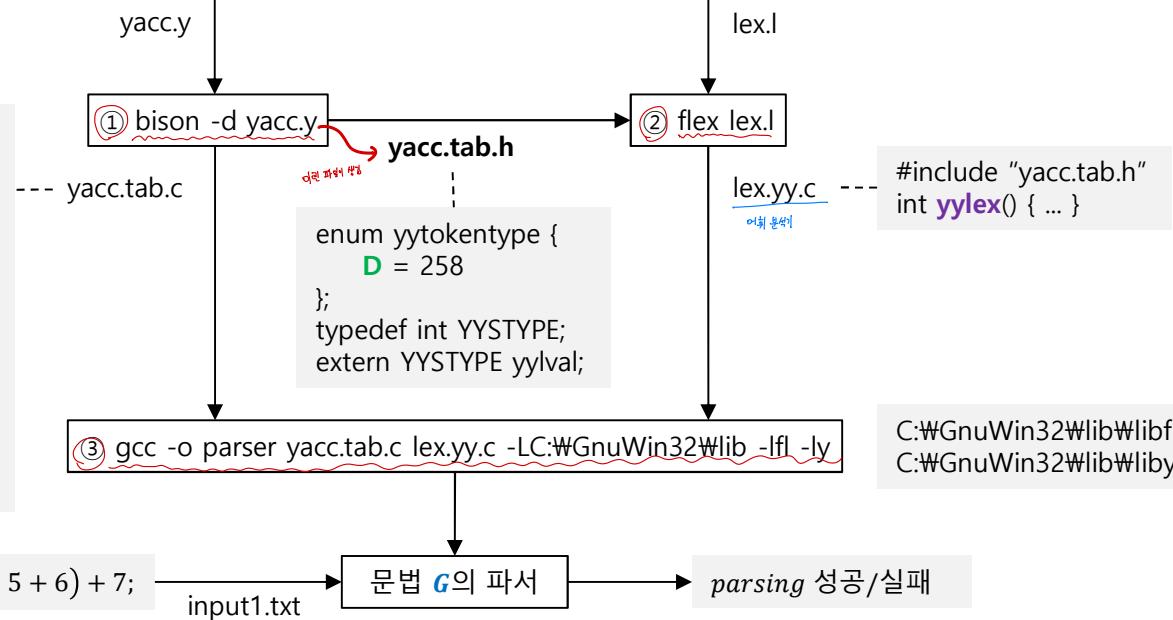
① bison -d yacc.y

② flex lex.l

③ gcc -o parser yacc.tab.c lex.yy.c -LC:\GnuWin32\lib -lfl -ly
 parser < input1.txt
 parsing: success

G 의 terminal symbol 구성 규칙

```
%
#include "yacc.tab.h" ← 실행 후 생성된 yacc.tab.h
%
[0-9] { return D; } ← 0 ~ 9가 들어온다 D로 반환
[+*()] { return yytext[0]; } ← + * ( )가 들어온다 문자로 반환
. { return yytext[0]; } ← .가 들어온다 문자로 반환
```



#3: 수식 문법과 파서 생성

파서 생성 및 파싱 예

```
bison -d yacc.y  
flex lex.l
```

```
gcc -o parser yacc.tab.c lex.yy.c -LC:\GnuWin32\lib -lfl -ly  
parser < input1.txt
```

F -> D

T -> F

E -> T

F -> D

T -> F

F -> D

T -> T * F

E -> E + T

S -> E

parsing: success

```
parser < input2.txt
```

F -> D

T -> F

E -> T

syntax error

parsing: fail

input1.txt

1+2*3;

input2.txt

12+34;

lex.l (lex 파일)

```
%{  
#include <stdio.h>  
#include "yacc.tab.h"  
%}  
  
%%  
[0-9] { return D; }  
[+*0] { return yytext[0]; }  
. { return yytext[0]; }  
%%
```

yacc.y (grammar 파일)

```
%{  
#include <stdio.h>  
int yylex();  
int yyerror(const char*);  
%}  
  
%token D  
  
%%  
S : E ';' { printf("S -> E\n"); }  
;  
E : E '+' T { printf("E -> E + T\n"); }  
| T { printf("E -> T\n"); }  
;  
T : T '*' F { printf("T -> T * F\n"); }  
| F { printf("T -> F\n"); }  
;  
F : '(' E ')' { printf("F -> ( E )\n"); }  
| D { printf("F -> D\n"); }  
;  
%%  
  
int main(){  
if(0==yyparse()) printf("parsing: success");  
else printf("parsing: fail");  
return 0;  
}
```

#4: 수식 문법과 파서 생성

lex.l (lex 파일)

```
%{  
#include <stdio.h>  
#include <stdlib.h>  
#include "yacc.tab.h"  
%}  
  
%%  
[0-9]+ { printf("<NUM,%s>\n", yytext); return NUM; }  
[;+*()]{ printf("<%s,%s>\n", yytext, yytext); return yytext[0]; }  
. { printf("<%s,%s>\n", yytext, yytext); return yytext[0]; }  
%%
```

파서 생성 및 파싱 예

```
bison -d yacc.y  
flex lex.l  
gcc -o parser yacc.tab.c lex.yy.c -LC:\GnuWin32\lib -lfl -ly  
parser < input2.txt
```

<NUM,12>

F -> NUM

T -> F

<+,+>

E -> T

<NUM,34>

F -> NUM

T -> F

<;>

E -> E + T

S -> E

parsing: success

input2.txt

12+34;

연습

- 뺄셈이 가능하도록 수정하시오
- 예) 1+2*(3-4);

yacc.y (grammar 파일)

```
%{  
#include <stdio.h>  
int yylex();  
int yyerror(const char*);  
%}  
  
%token NUM  
  
%%  
S : E ';' { printf("S -> E\n"); }  
| ;  
E : E '+' T { printf("E -> E + T\n"); }  
| T { printf("E -> T\n"); }  
| ;  
T : T '*' F { printf("T -> T * F\n"); }  
| F { printf("T -> F\n"); }  
| ;  
F : '(' E ')' { printf("F -> ( E )\n"); }  
| NUM { printf("F -> NUM\n"); }  
| ;  
%%  
  
int main(){  
    if(0==yparse()) printf("parsing: success");  
    else printf("parsing: fail");  
    return 0;  
}
```

yyval, \$\$, \$1, \$2, ...

문법 *G*

```
%token NUM
%%
S : E ';' { $$ = $1; printf("Result=%d\n", $$); }
E : E '+' T { $$ = $1 + $3; }
E : T { $$ = $1; }
T : T '*' F { $$ = $1 * $3; }
T : F { $$ = $1; }
F : '(' E ')'
F : NUM { $$ = $1; }
%%

```

```
enum yytokentype {
    NUM = 258
};
typedef int YYSTYPE;
YYSTYPE yyval;
...
int yyparse() {
    ... yylex()
}
int main() {
    ... yyparse()
}
```

- 토큰의 semantic value는 yyval(기본 타입은 int)에 저장
- \$\$ → 현재 rule의 LHS 심볼에 대한 semantic value
- \$n → 현재 rule의 RHS의 n번째 심볼에 대한 semantic value

*G*의 terminal symbol 구성 규칙

```
%
#include <stdlib.h>
#include "yacc.tab.h"
%
[0-9]+ { yyval=atoi(yytext); return NUM; }
[+*()] { return yytext[0]; }
.
```

askii to int
아스키 코드로 된 문자열을 int로 변환

yacc.y

bison -d yacc.y

yacc.tab.h

lex.l

flex lex.l

```
#include "yacc.tab.h"
int yylex() { ... }
```

--- yacc.tab.c

gcc -o parser yacc.tab.c lex.yy.c -LC:\GnuWin32\lib -lfl -ly

100 + 2 * (3 + 4);

input1.txt

문법 *G*의 파서

Result = 114

#5: 수식 문법과 파서 생성 (디폴트 yylval)

yacc.tab.h

```
enum yytokentype {  
    NUM = 258  
};  
typedef int YYSTYPE;  
extern YYSTYPE yylval;
```

파서 생성 및 파싱 예

```
parser < input1.txt  
F -> NUM,      100 -> 100  
T -> F,        100 -> 100  
E -> T,        100 -> 100  
F -> NUM,      2 -> 2  
T -> F,        2 -> 2  
F -> NUM,      3 -> 3  
T -> F,        3 -> 3  
E -> T,        3 -> 3  
F -> NUM,      4 -> 4  
T -> F,        4 -> 4  
E -> E + T,    7 -> 3 + 4  
F -> ( E ),   7 -> ( 7 )  
T -> T * F,   14 -> 2 * 7  
E -> E + T,   114 -> 100 + 14  
S -> E,        114 -> 114  
parsing: success
```

input1.txt

100+2*(3+4);

yacc.y (grammar 파일)

```
%{  
#include <stdio.h>  
int yylex();  
int yyerror(const char*);  
%}  
  
%token NUM
```

lex.l (lex 파일)

```
%{  
#include <stdlib.h>  
#include "yacc.tab.h"  
%}  
  
%%  
[0-9]+ { yylval=atoi(yytext); return NUM; }  
[;+*0] { return yytext[0]; }  
. { return yytext[0]; }  
%%
```

yytext[0]에 기본형을 반환할
그리고 그걸 NUM과 쓰면 될지.
그러나 yytext[0]은 다른걸 사용해도 되지 않을까?
디버깅

```
%%  
S : E ';' { $$ = $1; printf("S -> E, %d -> %d\n", $$, $1); }  
| ;  
E : E '+' T { $$ = $1 + $3; printf("E -> E + T, %d -> %d + %d\n", $$, $1, $3); }  
| T { $$ = $1; printf("E -> T, %d -> %d\n", $$, $1); }  
| ;  
T : T '*' F { $$ = $1 * $3; printf("T -> T * F, %d -> %d * %d\n", $$, $1, $3); }  
| F { $$ = $1; printf("T -> F, %d -> %d\n", $$, $1); }  
| ;  
F : '(' E ')' { $$ = $2; printf("F -> ( E ), %d -> ( %d )\n", $$, $2); }  
| NUM { $$ = $1; printf("F -> NUM, %d -> %d\n", $$, $1); }  
| ;  
%%  
int main(){  
    if(0==yparse()) printf("parsing: success");  
    else printf("parsing: fail");  
    return 0;  
}
```

#5A: 수식 문법과 파서 생성 (union 타입 yyval)

```
yacc.tab.h
enum yytokentype {
    NUM = 258
};
typedef union YYSTYPE {
    int intval;
} YYSTYPE;
extern YYSTYPE yyval;
```

파서 생성 및 파싱 예

```
parser < input1.txt
F -> NUM,      100 -> 100
T -> F,        100 -> 100
E -> T,        100 -> 100
F -> NUM,      2 -> 2
T -> F,        2 -> 2
F -> NUM,      3 -> 3
T -> F,        3 -> 3
E -> T,        3 -> 3
F -> NUM,      4 -> 4
T -> F,        4 -> 4
E -> E + T,    7 -> 3 + 4
F -> ( E ),   7 -> ( 7 )
T -> T * F,   14 -> 2 * 7
E -> E + T,   114 -> 100 + 14
S -> E,       114 -> 114
parsing: success
```

input1.txt
100+2*(3+4);

yacc.y (grammar 파일)

```
%{
#include <stdio.h>
#include <stdlib.h>
int yylex();
int yyerror(const char*);
%}

%union {
    int intval;
};

%token <intval> NUM
%type <intval> S E T F
```

lex.l (lex 파일)

```
%{
#include <stdlib.h>
#include "yacc.tab.h"
%}

%%

[0-9]+ { yyval.intval=atoi(yytext); return NUM; }
[;+*0] { return yytext[0]; }
.

%%
```

Int 타입은 그냥 바로 상관 없음을
그리고 YYSTYPE 타입으로 사용되며,
여기서는 그대로 코드를 작성해주세요.

```
%%
S : E ';'     { $$ = $1;           printf("S -> E, %d -> %d\n", $$, $1); }
;               ;
E : E '+' T  { $$ = $1 + $3;     printf("E -> E + T, %d -> %d + %d\n", $$, $1, $3); }
| T            { $$ = $1;           printf("E -> T, %d -> %d\n", $$, $1); }
;               ;
T : T '*' F  { $$ = $1 * $3;     printf("T -> T * F, %d -> %d * %d\n", $$, $1, $3); }
| F            { $$ = $1;           printf("T -> F, %d -> %d\n", $$, $1); }
;               ;
F : '(' E ')' { $$ = $2;           printf("F -> ( E ), %d -> ( %d )\n", $$, $2); }
| NUM          { $$ = $1;           printf("F -> NUM, %d -> %d\n", $$, $1); }
;

int main(){
    if(0==yparse()) printf("parsing: success");
    else printf("parsing: fail");
    return 0;
}
```

#5B: 수식 문법과 파서 생성 (union 타입 yyval)

yacc.tab.h

```
enum yytokentype {
    NUM = 258
};
typedef union YYSTYPE {
    double dblVal;
} YYSTYPE;
extern YYSTYPE yyval;
```

input1.txt

```
100+2*(3+4);
```

파서 생성 및 파싱 예

```
parser < input1.txt
F -> NUM, 100.00 -> 100.00
T -> F, 100.00 -> 100.00
E -> T, 100.00 -> 100.00
F -> NUM, 2.00 -> 2.00
T -> F, 2.00 -> 2.00
F -> NUM, 3.00 -> 3.00
T -> F, 3.00 -> 3.00
E -> T, 3.00 -> 3.00
F -> NUM, 4.00 -> 4.00
T -> F, 4.00 -> 4.00
E -> E + T, 7.00 -> 3.00 + 4.00
F -> ( E ), 7.00 -> ( 7.00 )
T -> T * F, 14.00 -> 2.00 * 7.00
E -> E + T, 114.00 -> 100.00 + 14.00
S -> E, 114.00 -> 114.00
parsing: success
```

yacc.y (grammar 파일)

```
%{
#include <stdio.h>
#include <stdlib.h>
int yylex();
int yyerror(const char*);
%}

%union {
    double dblVal;
};

%token <dblVal> NUM
%type <dblVal> S E T F
```

lex.l (lex 파일)

```
%{
#include <stdlib.h>
#include "yacc.tab.h"
%}

%%

[0-9]+ { yyval.dblVal=atof(yytext); return NUM; }

[+*0] { return yytext[0]; }

. { return yytext[0]; }

%%
```

```
%%

S : E ';' { $$ = $1; printf("S -> E, %.2lf -> %.2lf\n", $$, $1); }
| ; { $$ = $1; }

E : E '+' T { $$ = $1 + $3; printf("E -> E + T, %.2lf -> %.2lf + %.2lf\n", $$, $1, $3); }
| T { $$ = $1; printf("E -> T, %.2lf -> %.2lf\n", $$, $1); }

T : T '*' F { $$ = $1 * $3; printf("T -> T * F, %.2lf -> %.2lf * %.2lf\n", $$, $1, $3); }
| F { $$ = $1; printf("T -> F, %.2lf -> %.2lf\n", $$, $1); }

F : '(' E ')' { $$ = $2; printf("F -> ( E ), %.2lf -> ( %.2lf )\n", $$, $2); }
| NUM { $$ = $1; printf("F -> NUM, %.2lf -> %.2lf\n", $$, $1); }

%%

int main(){
    if(0==yparse()) printf("parsing: success");
    else printf("parsing: fail");
    return 0;
}
```

#5C: 수식 문법과 파서 생성 (union 타입 yyval)

yacc.tab.h

```
enum yytokentype {  
    NUM = 258  
};  
typedef union YYSTYPE {  
    char *strVal;  
    double dblVal;  
} YYSTYPE;  
extern YYSTYPE yyval;
```

input1.txt

```
100+2*(3+4);
```

파서 생성 및 파싱 예

```
parser < input1.txt  
F -> NUM, 100.00 -> 100  
T -> F, 100.00 -> 100.00  
E -> T, 100.00 -> 100.00  
F -> NUM, 2.00 -> 2  
T -> F, 2.00 -> 2.00  
F -> NUM, 3.00 -> 3  
T -> F, 3.00 -> 3.00  
E -> T, 3.00 -> 3.00  
F -> NUM, 4.00 -> 4  
T -> F, 4.00 -> 4.00  
E -> E + T, 7.00 -> 3.00 + 4.00  
F -> ( E ), 7.00 -> ( 7.00 )  
T -> T * F, 14.00 -> 2.00 * 7.00  
E -> E + T, 114.00 -> 100.00 + 14.00  
S -> E, 114.00 -> 114.00  
parsing: success
```

yacc.y (grammar 파일)

```
%{  
#include <stdio.h>  
#include <stdlib.h>  
int yylex();  
int yyerror(const char*);  
%}  
  
%union {  
    char *strVal;  
    double dblVal;  
};  
%token <strVal> NUM  
%type <dblVal> S E T F
```

```
%%  
S : E ';'      { $$ = $1; }  
; :  
E : E '+' T   { $$ = $1 + $3; }  
| T           { $$ = $1; }  
; :  
T : T '*' F   { $$ = $1 * $3; }  
| F           { $$ = $1; }  
; :  
F : '(' E ')' { $$ = $2; }  
| NUM         { $$ = atof($1); }  
; :  
%%  
int main(){  
    if(0==yparse()) printf("parsing: success");  
    else printf("parsing: fail");  
    return 0;  
}
```

lex.l (lex 파일)

```
%{  
#include <string.h>  
#include "yacc.tab.h"  
%}  
%%  
[0-9]+ { yyval.strVal=strdup(yytext); return NUM; }  
[;+*0] { return yytext[0]; }  
. { return yytext[0]; }  
%%
```

이 럭비로 문자는
숫자 용 가격 계산을 품으로 인식 학습하는 과정

```
printf("S -> E, %.2lf -> %.2lf\n", $$, $1); }  
printf("E -> E + T, %.2lf -> %.2lf + %.2lf\n", $$, $1, $3); }  
printf("E -> T, %.2lf -> %.2lf\n", $$, $1); }  
printf("T -> T * F, %.2lf -> %.2lf * %.2lf\n", $$, $1, $3); }  
printf("T -> F, %.2lf -> %.2lf\n", $$, $1); }  
printf("F -> ( E ), %.2lf -> ( %.2lf )\n", $$, $2); }  
printf("F -> NUM, %.2lf -> %s\n", $$, $1); }
```

위에서 숫자부분을 그대로 활용하니 예쁘게
답에서 처리해줄

다음장!

$S \rightarrow E ;$

$E \rightarrow E + T$

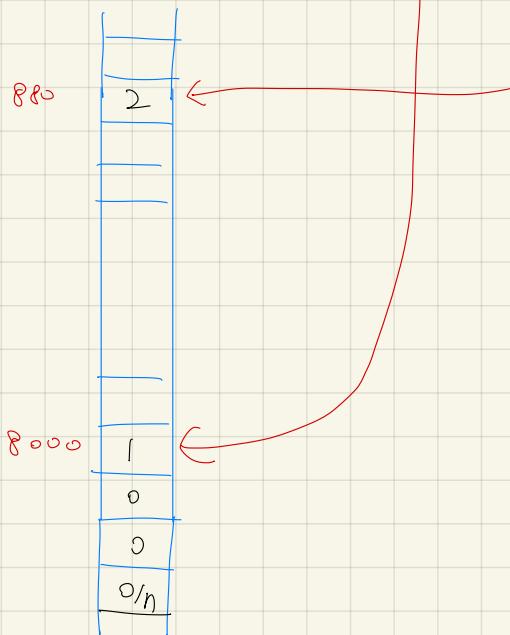
$E \rightarrow T$

$T \rightarrow T * F$

$T \rightarrow F$

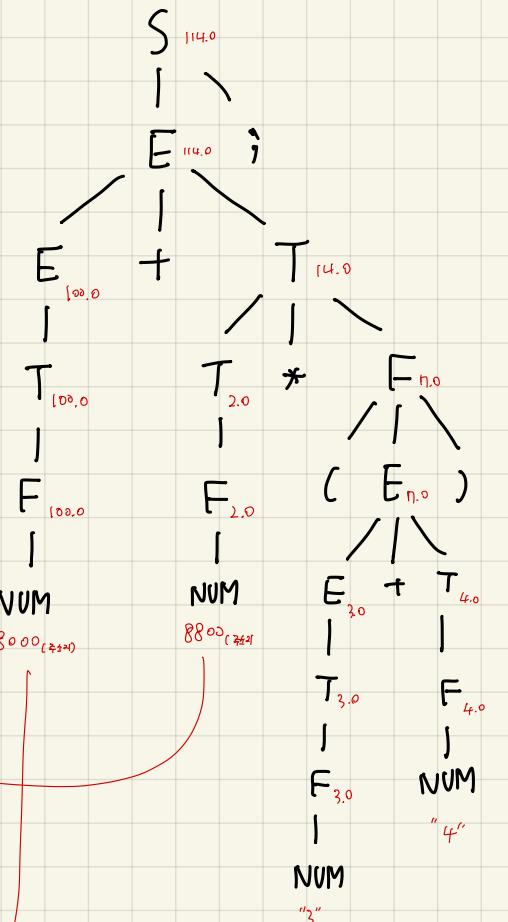
$F \rightarrow (E)$

$F \rightarrow \text{NUM}$



$100 + 2 * (3 + 4)$

累进



Parse < Input.txt

:

;

;

}

YY Parse()

YYlex()

T 언어

T 언어 grammar

프로그래밍언어 T

- 문장은 세미콜론으로 마침
- 숫자는 정수만 사용 가능
- 수식에서 괄호, +, * 연산자 사용 가능
- 변수명은 영어 소문자 하나로 정의
- 대입문, print문 사용 가능

$$\begin{aligned} Pgm &\rightarrow \text{DEF MAIN () } \{ \text{Stmts} \} \\ \text{Stmts} &\rightarrow \text{Stmt Stmt} \mid \epsilon \\ \text{Stmt} &\rightarrow ID = \text{Expr} ; \\ \text{Stmt} &\rightarrow \text{PRINT Expr} ; \\ \text{Expr} &\rightarrow \text{Expr} + \text{Term} \mid \text{Term} \\ \text{Term} &\rightarrow \text{Term} * \text{Factor} \mid \text{Factor} \\ \text{Factor} &\rightarrow (\text{Expr}) \mid \text{NUM} \mid \text{ID} \end{aligned}$$

T 언어 소스코드 #1

```
def main() {  
    x=3;  
    y=10+2*(x+1);  
    print x+y;  
}
```

T 언어 소스코드 #2

```
def main() {  
    x=3;  
    y=10+2*(x+1);  
    print x+y;  
    z=2;  
    print x+y+z;  
}
```

T 언어 소스코드 #3

```
def main() {  
    print x;  
    x=3;  
    y=10+2*(x+1);  
    print x+y;  
    z=2;  
    print x+y+z;  
}
```

#6: T 언어 파서

yacc.y	yacc.tab.h	int main(){ if(0==yparse()) printf("parsing: success"); else printf("parsing: fail"); return 0; }	Pgm → DEF MAIN () { Stmts } Stmts → Stmt Stmt ε Stmt → ID = Expr ; Stmt → PRINT Expr ; Expr → Expr + Term Term Term → Term * Factor Factor Factor → (Expr) NUM ID
		lex.l	
<pre>%{ #include <stdio.h> #include <stdlib.h> int yylex(); int yyerror(const char*); int var[26]; %} %union { char *strVal; int intValue; }; %token <strVal> NUM ID %type <intValue> Pgm Stmts Stmt Expr Term Factor %token MAIN PRINT DEF %% Pgm : DEF MAIN '(' ')' '{' Stmts '}' { } ; Stmts : Stmt Stmts { } ; Stmt : ID '=' Expr ';' → { var[\$1[0]-'a']=\$3; } PRINT Expr ';' { printf("%d\n", \$2); } ; Expr : Expr '+' Term { \$\$ = \$1 + \$3; } Term { \$\$ = \$1; } ; Term : Term '*' Factor { \$\$ = \$1 * \$3; } Factor { \$\$ = \$1; } ; Factor : '(' Expr ')' { \$\$ = \$2; } NUM { \$\$ = atoi(\$1); } ID { \$\$ = var[\$1[0]-'a']; } ; %%</pre>	<pre>enum yytokentype { NUM = 258, ID = 259, MAIN = 260, PRINT = 261, DEF = 262 }; typedef union YYSTYPE { char *strVal; int intValue; } YYSTYPE; extern YYSTYPE yylval;</pre>	<pre>int main(){ if(0==yparse()) printf("parsing: success"); else printf("parsing: fail"); return 0; }</pre>	$\text{Pgm} \rightarrow \text{DEF MAIN } () \{ \text{Stmts} \}$ $\text{Stmts} \rightarrow \text{Stmt Stmt} \mid \epsilon$ $\text{Stmt} \rightarrow \text{ID} = \text{Expr} ;$ $\text{Stmt} \rightarrow \text{PRINT Expr} ;$ $\text{Expr} \rightarrow \text{Expr} + \text{Term} \mid \text{Term}$ $\text{Term} \rightarrow \text{Term} * \text{Factor} \mid \text{Factor}$ $\text{Factor} \rightarrow (\text{Expr}) \mid \text{NUM} \mid \text{ID}$

T 언어 소스코드
input1.txt

```
def main() {  
    x=3;  
    y=10+2*(x+1);  
    print x+y;  
}
```

실행
parser < input1.txt
21
parsing: success

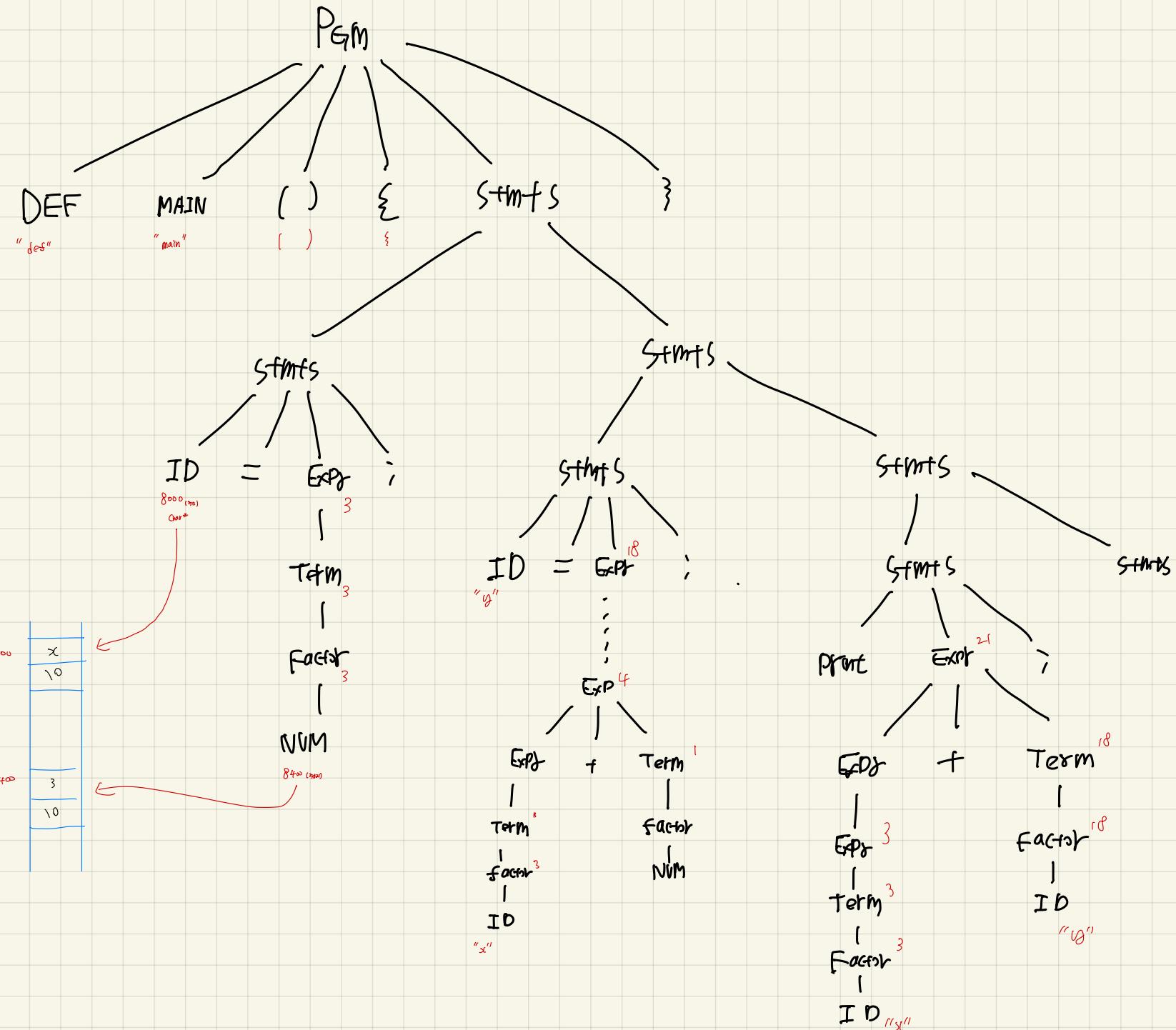
프로그래밍언어 T

- 문장은 세미콜론으로 마침
- 숫자는 정수만 사용 가능
- 수식에서 괄호, +, * 연산자 사용 가능
- 변수명은 영어 소문자 하나로 정의
- 대입문, print문 사용 가능

연습

- 뺄셈, 나눗셈이 가능하게 하시오 (나눗셈 결과는 정수로만 표현)
- 명령문 println;을 추가하시오 (println;은 C의 printf("\n");와 동일)

다음 강



#6A: T 언어 파서 (실수)

yacc.y

```
%{
#include <stdio.h>
#include <stdlib.h>
int yylex();
int yyerror(const char*);
double var[26];
%}

%union {
    char *strVal;
    double dblVal;
};

%token <strVal> NUM ID
%type <dblVal> Pgm Stmt Stmt Expr Term Factor
%token MAIN PRINT DEF

%%

Pgm : DEF MAIN '(' ')' '{' Stmt '}' { }

;

Stmt : Stmt Stmt { }
      | { }

;

Stmt : ID '=' Expr ';' { var[$1[0]-'a']=$3; }
      | PRINT Expr ';' { printf("%.2lf\n", $2); }

;

Expr : Expr '+' Term { $$ = $1 + $3; }
      | Expr '-' Term { $$ = $1 - $3; }
      | Term { $$ = $1; }

;

Term : Term '*' Factor { $$ = $1 * $3; }
      | Term '/' Factor { $$ = $1 / $3; }
      | Factor { $$ = $1; }

;

Factor : '(' Expr ')'
        | NUM { $$ = atof($1); }
        | ID { $$ = var[$1[0]-'a']; }

;

%%
```

실수에게 Double를 사용함

```
int main(){
    if(0==yparse()) printf("parsing: success");
    else printf("parsing: fail");
}
```

lex.l

```
%{
#include <string.h>
#include "yacc.tab.h"
%}

%%

"def" { return DEF; }
"main" { return MAIN; }
"print" { return PRINT; }
[0-9]+(\.[0-9]*)? { yylval.strVal=strdup(yytext); return NUM; }
[a-z] { yylval.strVal=strdup(yytext); return ID; }
[ \t\n] { }
[-{};+*()] { return yytext[0]; }
. { return yytext[0]; }

%%
```

input1.txt

```
def main() {
    x=1.0;
    y=2;
    z=3;
    w=4;
    v=0-5;
    print x+y-z*w/v;
}
```

실행

```
parser < input1.txt
5.40
parsing: success
```

#6B: T 언어 파서

연산자 우선순위(precedence) 및 결합규칙(associativity) 선언

yacc.y

```
%{
#include <stdio.h>
#include <stdlib.h>
int yylex();
int yyerror(const char*); 
double var[26];
%}

%union {
    char *strVal;
    double dblVal;
};

%token <strVal> NUM ID
%type <dblVal> Pgm Stmt Stmt Expr
%token MAIN PRINT DEF

%left '+' '-'
%left '*' '/'

%%

Pgm : DEF MAIN '(' ')' '{' Stmt '}' { }

;

Stmts : Stmt Stmt { }
| { }

;

Stmt : ID '=' Expr ';' { var[$1[0]-'a']=$3; }
| PRINT Expr ';' { printf("%.2lf\n", $2); }

;

Expr : Expr '+' Expr { $$ = $1 + $3; }
| Expr '-' Expr { $$ = $1 - $3; }
| Expr '*' Expr { $$ = $1 * $3; }
| Expr '/' Expr { $$ = $1 / $3; }
| '(' Expr ')'
| NUM { $$ = atof($1); }
| ID { $$ = var[$1[0]-'a']; }

;

%%
```

```
int main(){
    if(0==yparse()) printf("parsing: success");
    else printf("parsing: fail");
    return 0;
}
```

lex.l

```
%{
#include <string.h>
#include "yacc.tab.h"
%}

%%

"def" { return DEF; }
"main" { return MAIN; }
"print" { return PRINT; }
[0-9]+(\.[0-9]*)? { yyval.strVal=strdup(yytext); return NUM; }
[a-z]
[ \t\n]
[-{};+*()]
.

%%
```

input1.txt

```
def main() {
    x=1.0;
    y=2.;
    z=3;
    w=4;
    v=0.5;
    print x+y-z*w/v;
}
```

실행

```
parser < input1.txt
5.40
parsing: success
```

T 언어 (while문 추가)

프로그래밍언어 T

- 문장은 세미콜론으로 마침
- 숫자는 정수만 사용 가능
- 수식에서 괄호, +, * 연산자 사용 가능
- 변수명은 영어 소문자 하나로 정의
- 대입문, print문, while문 사용 가능

T 언어 grammar

$$\begin{aligned} Pgm &\rightarrow \text{DEF MAIN () } \{ \text{Stmts} \} \\ \text{Stmts} &\rightarrow \text{Stmt Stmt} \mid \epsilon \\ \text{Stmt} &\rightarrow ID = \text{Expr} ; \\ \text{Stmt} &\rightarrow \text{PRINT Expr} ; \\ \text{Stmt} &\rightarrow \text{WHILE (Expr) Stmt} ; \\ \text{Stmt} &\rightarrow \{ \text{Stmts} \} \end{aligned}$$
$$\begin{aligned} \text{Expr} &\rightarrow \text{Expr} + \text{Term} \mid \text{Term} \\ \text{Term} &\rightarrow \text{Term} * \text{Factor} \mid \text{Factor} \\ \text{Factor} &\rightarrow (\text{Expr}) \mid \text{NUM} \mid ID \end{aligned}$$

#7: T 언어 파서: 추상구문트리 생성

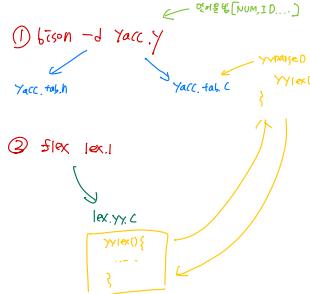
yacc.y

```
%{
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
int yylex();
int yyerror(const char*);
double var[26];

struct Node {
    int nType;
    char *lexeme;
    struct Node *c1;
    struct Node *c2;
};

enum AST_Node_Type {
    nSTMTS, nASSIGN, nPRINT, nWHILE,
    nADD, nSUB, nMUL, nDIV, nNUM,
    nID
};
char *nTypeName[]={
    "nSTMTS", "nASSIGN", "nPRINT",
    "nWHILE", "nADD", "nSUB", "nMUL",
    "nDIV", "nNUM", "nID"
};

struct Node *ast;
struct Node *makeNode(int nType,
    struct Node *c1, struct Node *c2);
%}
```



```
%union {
    char *strVal;
    struct Node *node;
};

%token <strVal> NUM ID
%type <node> Pgm Stmt Stmt Expr Term Factor
%token MAIN PRINT WHILE DEF

%%
Pgm   : DEF MAIN '(' ')' '{' Stmt '}'      { ast = $6; }
;
Stmt  : Stmt Stmt                     { $$ = makeNode(nSTMTS,$1,$2); }
|                                { $$ = NULL; }
;
Stmt  : ID '=' Expr ';'              {
    struct Node *t=makeNode(nID,NULL,NULL);
    t->lexeme=strdup($1);
    $$ = makeNode(nASSIGN,t,$3);
}
| PRINT Expr ';'
| WHILE '(' Expr ')' Stmt
| '{' Stmt '}'                   { $$ = $2; }
;
Expr  : Expr '+' Term             { $$ = makeNode(nADD,$1,$3); }
| Expr '-' Term               { $$ = makeNode(nSUB,$1,$3); }
| Term                         { $$ = $1; }
;
Term   : Term '*' Factor          { $$ = makeNode(nMUL,$1,$3); }
| Term '/' Factor            { $$ = makeNode(nDIV,$1,$3); }
| Factor                      { $$ = $1; }
;
Factor : '(' Expr ')'
| NUM                         { $$ = $2; }
;
%%
```

```
struct Node *makeNode(int nType, struct Node *c1, struct Node *c2){
    struct Node *p=(struct Node *)malloc(sizeof(struct Node));
    if(p==NULL) { yyerror("memory error"); exit(-1); }
    p->nType=nType;
    p->c1=c1;
    p->c2=c2;
    return p;
}

void postorder(struct Node *p){
    if(p==NULL) return;
    postorder(p->c1);
    postorder(p->c2);
    if(p->nType==nNUM) printf("NUM (%s)\n", p->lexeme);
    else if(p->nType==nID) printf("ID (%s)\n", p->lexeme);
    else printf("%s\n", nTypeName[p->nType]);
}

int main(){
    if(yyparse()) {
        printf("parsing: success\n");
        postorder(ast);
    }
    else printf("parsing: fail\n");
    return 0;
}
```

lex.l

```
%{
#include <string.h>
#include "yacc.tab.h"
%}

%%
"def"           { return DEF; }
"main"          { return MAIN; }
"print"         { return PRINT; }
"while"         { return WHILE; }
[0-9]+(\.[0-9]*?)? { yyval.strVal=strdup(yytext); return NUM; }
[a-z]           { yyval.strVal=strdup(yytext); return ID; }
[ \t\n]          { }
[-+=/*]          { return yytext[0]; }
.                { return yytext[0]; }
%%
```

#7: T 언어 파서: 추상구문트리 생성

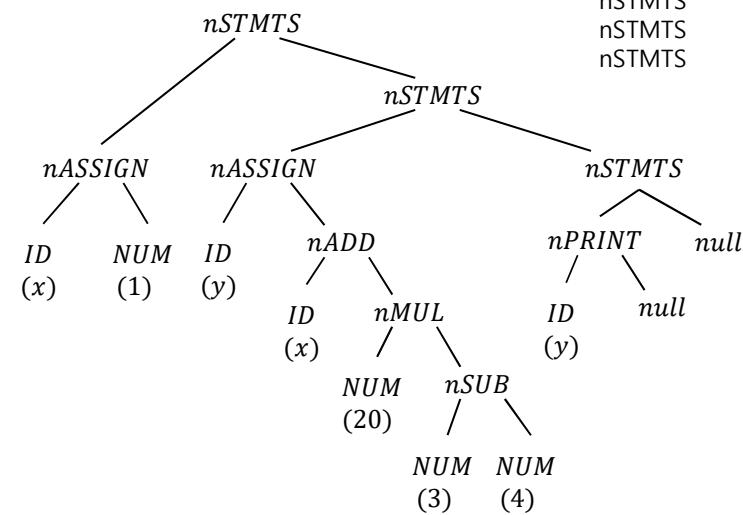
```
%%
Pgm  : DEF MAIN '(' '{' Stmts '}'      { ast = $6; }
;
Stmts : Stmt Stmt           { $$ = makeNode(nSTMTS,$1,$2); }
|      { $$ = NULL; }
;
Stmt  : ID '=' Expr ';'    {
                           struct Node *t=makeNode(nID,NULL,NULL);
                           t->lexeme=strdup($1);
                           $$ = makeNode(nASSIGN,t,$3);
                         }
| PRINT Expr ';'          { $$ = makeNode(nPRINT,$2,NULL); }
| WHILE '(' Expr ')' Stmt { $$ = makeNode(nWHILE,$3,$5); }
| '{' Stmts '}'          { $$ = $2; }
;
Expr   : Expr '+' Term     { $$ = makeNode(nADD,$1,$3); }
| Expr '-' Term          { $$ = makeNode(nSUB,$1,$3); }
| Term                  { $$ = $1; }
;
Term   : Term '*' Factor  { $$ = makeNode(nMUL,$1,$3); }
| Term '/' Factor        { $$ = makeNode(nDIV,$1,$3); }
| Factor                 { $$ = $1; }
;
Factor : '(' Expr ')'     { $$ = $2; }
| NUM                   {
                           $$ = makeNode(nNUM,NULL,NULL);
                           $$->lexeme=strdup($1);
                         }
| ID                     {
                           $$ = makeNode(nID,NULL,NULL);
                           $$->lexeme=strdup($1);
                         }
;
%%
```

input1.txt

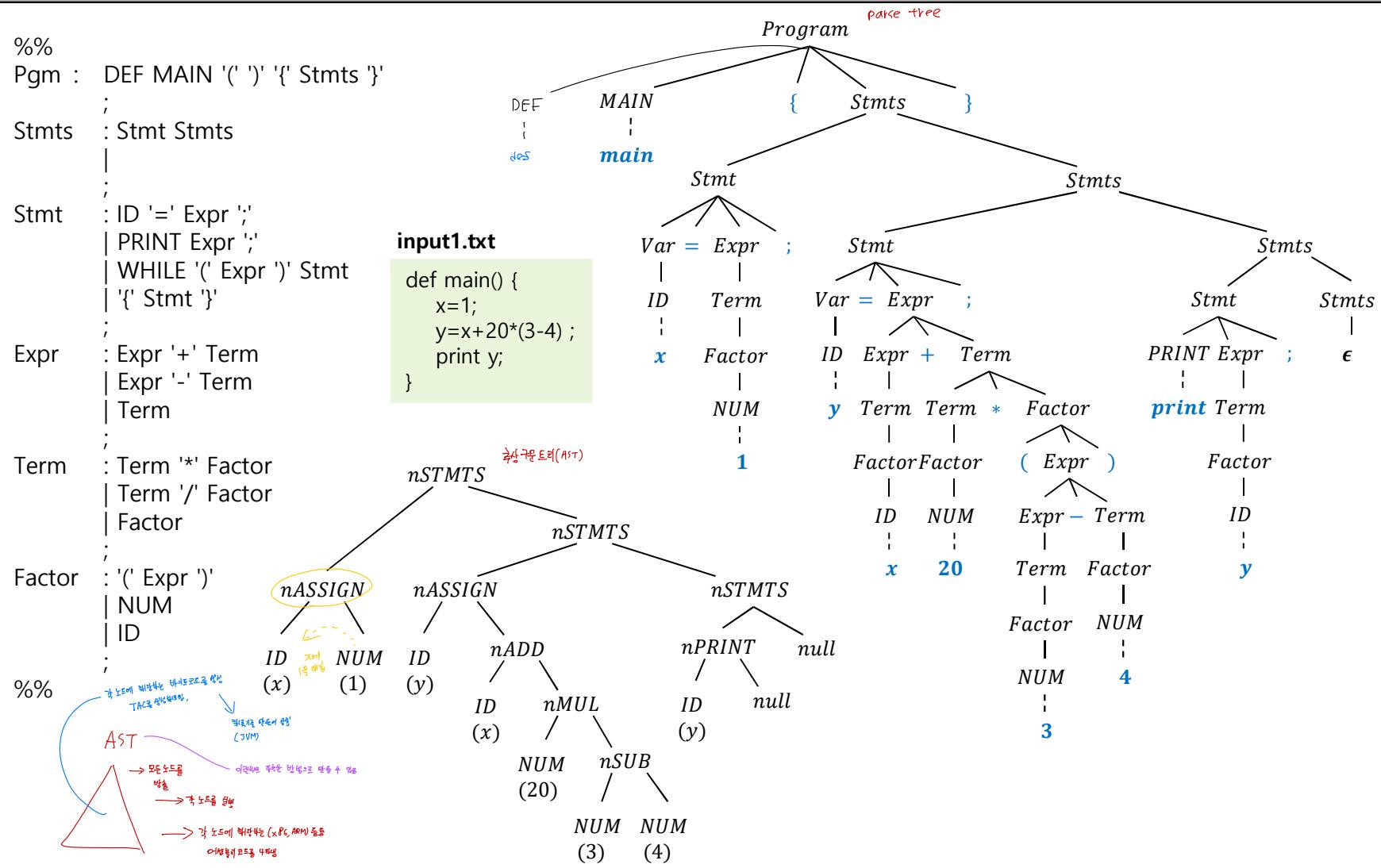
```
def main() {
    x=1;
    y=x+20*(3-4) ;
    print y;
}
```

실행

```
parser < input1.txt
parsing: success
ID (x)
NUM (1)
nASSIGN
ID (y)
ID (x)
NUM (20)
NUM (3)
NUM (4)
nSUB
nMUL
nADD
nASSIGN
ID (y)
nPRINT
nSTMTS
nSTMTS
nSTMTS
```



#7: T 언어 파서: 파스트리 vs. 추상구문트리



#8: T 언어 해석기 (1/2)

yacc.y

```
%{
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
int yylex();
int yyerror(const char*);
struct Node {
    int nType;
    double value;
    char *lexeme;
    struct Node *c1, *c2;
};
enum AST_Node_Type { nSTMTS, nASSIGN, nPRINT, nWHILE,
nADD, nSUB, nMUL, nDIV, nNUM, nID };
char *nTypeName[]={"nSTMTS", "nASSIGN", "nPRINT",
"nWHILE", "nADD", "nSUB", "nMUL", "nDIV", "nNUM", "nID"};
struct Node *makeNode(int nType, struct Node *c1, struct Node *c2);
struct Node *ast;
double var[26];
int ref[26]={0,};
%}
```

lex.l

```
%{
#include <string.h>
#include "yacc.tab.h"
%}

%%
"def"      { return DEF; }
"main"     { return MAIN; }
"print"    { return PRINT; }
"while"   { return WHILE; }
[0-9]+([.][0-9]*)>? { yyval.strVal=strdup(yytext); return NUM; }
[a-z]      { yyval.strVal=strdup(yytext); return ID; }
[ \t\n]     { }
[-;+/*]   { return yytext[0]; }
[.]       { return yytext[0]; }
%%
```

```
%union {
    char *strVal;
    struct Node *node;
};

%token <strVal> NUM ID
%type <node> Pgm Stmt Stmt Expr Term Factor
%token MAIN PRINT WHILE DEF

%%
Pgm : DEF MAIN '(' ')' '{' Stmts '}'      { ast = $6; }
      ;
Stmts : Stmt Stmt           { $$ = makeNode(nSTMTS,$1,$2); }
        |                   { $$ = NULL; }

Stmt : ID '=' Expr ';'          {
        struct Node *t=makeNode(nID,NULL,NULL);
        t->lexeme=strdup($1);
        $$ = makeNode(nASSIGN,t,$3);
    }
      | PRINT Expr ';'        { $$ = makeNode(nPRINT,$2,NULL); }
      | WHILE '(' Expr ')' Stmt { $$ = makeNode(nWHILE,$3,$5); }
      | '{' Stmts '}'         { $$ = $2; }

Expr : Expr '+' Term          { $$ = makeNode(nADD,$1,$3); }
      | Expr '-' Term          { $$ = makeNode(nSUB,$1,$3); }
      | Term                  { $$ = $1; }

Term : Term '*' Factor        { $$ = makeNode(nMUL,$1,$3); }
      | Term '/' Factor        { $$ = makeNode(nDIV,$1,$3); }
      | Factor                 { $$ = $1; }

Factor : '(' Expr ')'         { $$ = $2; }
        | NUM                  {
            $$ = makeNode(nNUM,NULL,NULL);
            $$->lexeme=strdup($1);
        }
        | ID                   {
            $$ = makeNode(nID,NULL,NULL);
            $$->lexeme=strdup($1);
        }

%%
```

void set(char *lexeme, double value){
 var[lexeme[0]-‘a’]=value;
 ref[lexeme[0]-‘a’]=1;
}
double get(char *lexeme){
 if(ref[lexeme[0]-‘a’]>0) return var[lexeme[0]-‘a’];
 printf("%s: uninitialized\n", lexeme);
 exit(-1);
}
double eval(struct Node *p){
 if(p==NULL) return 0;
 if(p->nType==nSTMTS) { eval(p->c1); eval(p->c2); return 0; }
 if(p->nType==nASSIGN) { set(p->c1->lexeme,eval(p->c2)); return 0; }
 if(p->nType==nPRINT) { printf("%2lf", eval(p->c1)); return 0; }
 if(p->nType==nWHILE) { while(eval(p->c1)) eval(p->c2); return 0; }
 if(p->nType==nADD) { return eval(p->c1) + eval(p->c2); }
 if(p->nType==nSUB) { return eval(p->c1) - eval(p->c2); }
 if(p->nType==nMUL) { return eval(p->c1) * eval(p->c2); }
 if(p->nType==nDIV) { return eval(p->c1) / eval(p->c2); }
 if(p->nType==nNUM) { p->value=atof(p->lexeme); return p->value; }
 if(p->nType==nID) { return get(p->lexeme); }
 return 0;
}
struct Node *makeNode(int nType, struct Node *c1, struct Node *c2){
 struct Node *p=(struct Node *)malloc(sizeof(struct Node));
 if(p==NULL) { printf("memory error"); exit(-1); }
 p->nType=nType;
 p->c1=c1;
 p->c2=c2;
 return p;
}
void postorder(struct Node *p){
 if(p==NULL) return;
 postorder(p->c1);
 postorder(p->c2);
 if(p->nType==nNUM) printf("NUM (%s)\n", p->lexeme);
 else if(p->nType==nID) printf("ID (%s)\n", p->lexeme);
 else printf("%s\n", nTypeName[p->nType]);
}
int main(){
 if(!yparse()) {
 printf("parsing: success\n");
 postorder(ast);
 eval(ast);
 }
 else printf("parsing: fail\n");
 return 0;
}

파란색 원에
ast에 반올림 해석 결과

#8: T 언어 해석기 (2/2)

```
void set(char *lexeme, double value){
    var[lexeme[0]-'a']=value;
    ref[lexeme[0]-'a']=1;
}
```

```
double get(char *lexeme){
    if(ref[lexeme[0]-'a']>0) return var[lexeme[0]-'a'];
    printf("%s: uninitialized\n", lexeme);
    exit(-1);
}
```

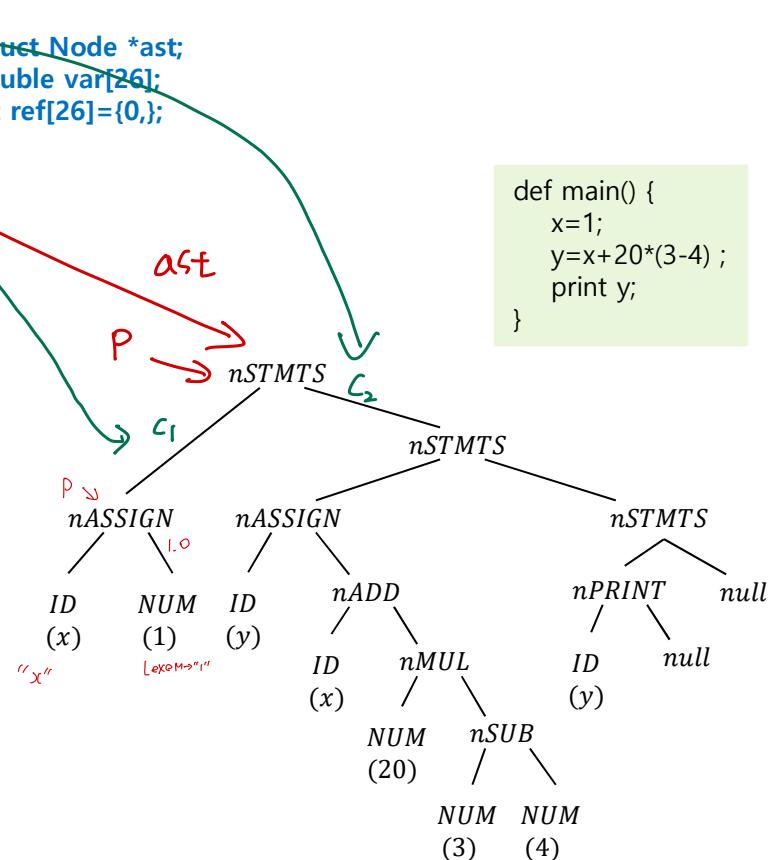
```
double eval(struct Node *p){
    if(p==NULL) return 0;
    if(p->nType==nSTMTS) { eval(p->c1); eval(p->c2); return 0; }
    if(p->nType==nASSIGN) { set(p->c1->lexeme,eval(p->c2)); return 0; }
    if(p->nType==nPRINT) { printf("%.2lf", eval(p->c1)); return 0; }
    if(p->nType==nWHILE) { while(eval(p->c1)) eval(p->c2); return 0; }
    if(p->nType==nADD) { return eval(p->c1) + eval(p->c2); }
    if(p->nType==nSUB) { return eval(p->c1) - eval(p->c2); }
    if(p->nType==nMUL) { return eval(p->c1) * eval(p->c2); }
    if(p->nType==nDIV) { return eval(p->c1) / eval(p->c2); }
    if(p->nType==nNUM) { p->value=atof(p->lexeme); return p->value; }
    if(p->nType==nID) { return get(p->lexeme); }
    return 0;
}
```

```
int main(){
    if(!yparse()) {
        printf("parsing: success\n");
        postorder(ast);
        eval(ast);
    }
    else printf("parsing: fail\n");
    return 0;
}
```

```
struct Node {
    int nType;
    double value;
    char *lexeme;
    struct Node *c1, *c2;
};

struct Node *ast;
double var[26];
int ref[26]={0,};
```

```
def main() {
    x=1;
    y=x+20*(3-4);
    print y;
}
```



#9: T 언어 (TAC 생성)

```
%{
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
int yylex();
int yyerror(const char*);
struct Node {
    int nType, tNo, INo1, INo2;
    double value;
    char *lexeme;
    struct Node *c1, *c2;
};
enum AST_Node_Type { nSTMTS, nASSIGN, nPRINT, nWHILE, nADD, nSUB, nMUL,
nDIV, nNUM, nID };
struct Node *makeNode(int nType, struct Node *c1, struct Node *c2);
struct Node *ast;
int INo=0, tNo=0;
%}

%union {
    char *strVal;
    struct Node *node;
};

%token <strVal> NUM ID
%type <node> Pgm Stmt Stmt Expr Term Factor
%token MAIN PRINT WHILE DEF

%%
Pgm   : DEF MAIN '(' ')' '{'Stmts'}'      { ast = $6; }
;
Stmts : Stmt Stmt           { $$ = makeNode(nSTMTS,$1,$2); }
|      { $$ = NULL; }
;
Stmt : ID '=' Expr ';'     { struct Node *t=makeNode(nID,NULL,NULL);
                            t->lexeme=strdup($1);
                            $$ = makeNode(nASSIGN,t,$3); }
| PRINT Expr ';'           { $$ = makeNode(nPRINT,$2,NULL); }
| WHILE '(' Expr ')' Stmt { $$ = makeNode(nWHILE,$3,$5); }
| '{'Stmts'}'              { $$ = $2; }
;
Expr : Expr '+' Term       { $$ = makeNode(nADD,$1,$3); }
| Expr '-' Term           { $$ = makeNode(nSUB,$1,$3); }
| Term                   { $$ = $1; }
;
Term: Term '*' Factor     { $$ = makeNode(nMUL,$1,$3); }
| Term '/' Factor         { $$ = makeNode(nDIV,$1,$3); }
| Factor                  { $$ = $1; }
;
Factor : '(' Expr ')'     { $$ = $2; }
| NUM                     { $$ = makeNode(nNUM,NULL,NULL);
                            $$->lexeme=strdup($1); }
| ID                      { $$ = makeNode(nID,NULL,NULL);
                            $$->lexeme=strdup($1); }
;
%%
```

```
void cg(struct Node *p){
    if(p==NULL) return;
    if(p->nType==nSTMTS) { cg(p->c1); cg(p->c2); return; }
    if(p->nType==nASSIGN) { cg(p->c2); printf("%s = %d\n", p->c1->lexeme, p->c2->tNo); return; }
    if(p->nType==nPRINT) {
        cg(p->c1);
        printf("param %d\n", p->c1->tNo);
        printf("call print\n");
        return;
    }
    if(p->nType==nWHILE) {
        p->INo1=++INo;
        p->INo2=++INo;
        printf("L%d:\n", p->INo1);
        cg(p->c1);
        printf("iffalse %d goto L%d\n", p->c1->tNo, p->INo2);
        cg(p->c2);
        printf("goto L%d\n", p->INo1);
        printf("L%d:\n", p->INo2);
        return;
    }
    if(p->nType==nADD) { cg(p->c1); cg(p->c2); p->tNo=++tNo; printf("%d = %d + %d\n", p->tNo, p->c1->tNo, p->c2->tNo); return; }
    if(p->nType==nSUB) { cg(p->c1); cg(p->c2); p->tNo=++tNo; printf("%d = %d - %d\n", p->tNo, p->c1->tNo, p->c2->tNo); return; }
    if(p->nType==nMUL) { cg(p->c1); cg(p->c2); p->tNo=++tNo; printf("%d = %d * %d\n", p->tNo, p->c1->tNo, p->c2->tNo); return; }
    if(p->nType==nDIV) { cg(p->c1); cg(p->c2); p->tNo=++tNo; printf("%d = %d / %d\n", p->tNo, p->c1->tNo, p->c2->tNo); return; }
    if(p->nType==nID) { p->tNo=++tNo; printf("t%d = %s\n", p->tNo,p->lexeme); return; }
    if(p->nType==nNUM) { p->tNo=++tNo; printf("t%d = %s\n", p->tNo,p->lexeme); return; }
}

struct Node *makeNode(int nType, struct Node *c1, struct Node *c2){
    struct Node *p=(struct Node *)malloc(sizeof(struct Node));
    if(p==NULL) { printf("memory error"); exit(-1); }
    p->nType=nType;
    p->c1=c1;
    p->c2=c2;
    return p;
}

int main(){
    if(!yparse()) {
        printf("parsing: success\n");
        cg(ast);
    }
    else printf("parsing: fail\n");
    return 0;
}
```

yacc.y

```
%{
#include <string.h>
#include "yacc.tab.h"
%}

%%

"def"          { return DEF; }
"main"         { return MAIN; }
"print"        { return PRINT; }
"while"        { return WHILE; }
[0-9]+(\.[0-9]+)? { yyval.strVal=strdup(yytext); return NUM; }
[a-z]          { yyval.strVal=strdup(yytext); return ID; }
[ \t\n]          { }
[-{};+*()]{ return yytext[0]; }
.               { return yytext[0]; }

%%
```

lex.l

#9: T 언어 (TAC 생성)

```
struct Node {
    int nType, tNo, INo1, INo2;
    double value;
    char *lexeme;
    struct Node *c1, *c2;
};
int INo=0, tNo=0;
```

```
void cg(struct Node *p){
    if(p==NULL) return;
    if(p->nType==nSTMTS) { cg(p->c1); cg(p->c2); return; }
    if(p->nType==nASSIGN) { cg(p->c2); printf("%s = %d\n", p->c1->lexeme, p->c2->tNo); return; }
    if(p->nType==nPRINT) {
        cg(p->c1);
        printf("param %d\n", p->c1->tNo);
        printf("call print\n");
        return;
    }
    if(p->nType==nWHILE) {
        p->INo1=++INo;
        p->INo2=++INo;
        printf("L%d:\n", p->INo1);
        cg(p->c1);
        printf("iffalse %d goto L%d\n", p->c1->tNo, p->INo2);
        cg(p->c2);
        printf("goto L%d\n", p->INo1);
        printf("L%d:\n", p->INo2);
        return;
    }
    if(p->nType==nADD) { cg(p->c1); cg(p->c2); p->tNo=++tNo; printf("t%d = t%d + t%d\n", p->tNo, p->c1->tNo, p->c2->tNo); return; }
    if(p->nType==nSUB) { cg(p->c1); cg(p->c2); p->tNo=++tNo; printf("t%d = t%d - t%d\n", p->tNo, p->c1->tNo, p->c2->tNo); return; }
    if(p->nType==nMUL) { cg(p->c1); cg(p->c2); p->tNo=++tNo; printf("t%d = t%d * t%d\n", p->tNo, p->c1->tNo, p->c2->tNo); return; }
    if(p->nType==nDIV) { cg(p->c1); cg(p->c2); p->tNo=++tNo; printf("t%d = t%d / t%d\n", p->tNo, p->c1->tNo, p->c2->tNo); return; }
    if(p->nType==nNUM) { p->tNo=++tNo; printf("t%d = %s\n", p->tNo, p->lexeme); return; }
    if(p->nType==nID) { p->tNo=++tNo; printf("t%d = %s\n", p->tNo, p->lexeme); return; }
}
```

input2.txt

```
def main() {
    n=10;
    s=0;
    while(n){
        s=s+n;
        n=n-1;
    }
    print s;
}
```

실행

parser < input2.txt
 parsing: success
 t1 = 10
 n = t1
 t2 = 0
 s = t2
 L1:
 t3 = n
 ifffalse t3 goto L2
 t4 = s
 t5 = n
 t6 = t4 + t5
 s = t6
 t7 = n
 t8 = 1
 t9 = t7 - t8
 n = t9
 goto L1
 L2:
 t10 = s
 param t10
 call print

#9: T 언어 (컴파일러, x86)

yacc.y

```

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
int yylex();
int yyerror(const char*);
```

struct Node {

- int nType, tNo, INo1, INo2;
- double value;
- char *lexeme;
- struct Node *c1, *c2;

};

enum AST_Node_Type { nSTMTS, nASSIGN, nPRINT, nWHILE, nADD,

nSUB, nMUL, nDIV, nNUM, nID };

struct Node *makeNode(int nType, struct Node *c1, struct Node *c2);

struct Node *ast;

double var[26];

int ref[26]={0,};

int INo=0, tNo=0;

%}

%union {

- char *strVal;
- struct Node *node;

};

%token <strVal> NUM ID

%type <node> Pgm Stmt Stmt Expr Term Factor

%token MAIN PRINT WHILE DEF

%%

Pgm : DEF MAIN '(' ')' '{' '}' Stmts ; { ast = \$6; }

;

Stmts : Stmt Stmt ; { \$\$ = makeNode(nSTMTS,\$1,\$2); }

| ; { \$\$ = NULL; }

;

Stmt : ID '=' Expr ; {

- struct Node *t=makeNode(nID,NULL,NULL);
- t->lexeme=strdup(\$1);
- \$\$ = makeNode(nASSIGN,t,\$3);

| PRINT Expr ; { \$\$ = makeNode(nPRINT,\$2,NULL); }

| WHILE '(' Expr ')' Stmt ; { \$\$ = makeNode(nWHILE,\$3,\$5); }

| '(' Stmts ')' { \$\$ = \$2; }

;

Expr : Expr '+' Term ; { \$\$ = makeNode(nADD,\$1,\$3); }

| Expr '-' Term ; { \$\$ = makeNode(nSUB,\$1,\$3); }

| Term ; { \$\$ = \$1; }

;

Term : Term '*' Factor ; { \$\$ = makeNode(nMUL,\$1,\$3); }

| Term '/' Factor ; { \$\$ = makeNode(nDIV,\$1,\$3); }

| Factor ; { \$\$ = \$1; }

;

Factor: '(' Expr ')' ; { \$\$ = \$2; }

| NUM ; { \$\$ = makeNode(nNUM,NULL,NULL); }

| ID ; {

- struct Node *t=makeNode(nID,NULL,NULL);
- t->lexeme=strdup(\$1);

}

;

```

void set(char *lexeme, double value){
    var[lexeme[0] - 'a'] = value;
    ref[lexeme[0] - 'a'] = 1;
}
double get(char *lexeme){
    if(ref[lexeme[0] - 'a'] > 0) return var[lexeme[0] - 'a'];
    printf("%s: uninitialized\n", lexeme);
    exit(-1);
}
void cg(struct Node *p){
    if(p == NULL) return;
    if(p->nType == nSTMTS) { cg(p->c1); cg(p->c2); return; }
    if(p->nType == nASSIGN) { cg(p->c2); set(p->c1->lexeme, 1); printf("\t\tmov eax, [%d]\n\t\tmov [%s], eax\n", p->c2->tNo, p->c1->lexeme); return; }
    if(p->nType == nPRINT) {
        cg(p->c1);
        printf("\t\tmov eax, [%d]\n\t\tpush eax\n\t\tpush formatStr\n", p->c1->tNo);
        printf("\t\tcall _printf\n\t\tadd esp, 8\n");
        return;
    }
    if(p->nType == nWHILE) {
        p->INo1 = ++INo;
        p->INo2 = ++INo;
        printf("L%d:\n", p->INo1);
        cg(p->c1);
        printf("\t\tmov eax, [%d]\n\t\tcmp eax, 0\n\t\tje L%d\n", p->c1->tNo, p->INo2);
        cg(p->c2);
        printf("\t\tjmp L%d\n", p->INo1);
        printf("L%d:\n", p->INo2);
        return;
    }
    if(p->nType == nADD) { cg(p->c1); cg(p->c2); p->tNo = ++tNo; printf("\t\tmov eax, [%d]\n\t\tadd eax, [%d]\n\t\tmov [%d], eax\n", p->c1->tNo, p->c2->tNo, p->tNo); return; }
    if(p->nType == nSUB) { cg(p->c1); cg(p->c2); p->tNo = ++tNo; printf("\t\tmov eax, [%d]\n\t\tsub eax, [%d]\n\t\tmov [%d], eax\n", p->c1->tNo, p->c2->tNo, p->tNo); return; }
    if(p->nType == nMUL) { cg(p->c1); cg(p->c2); p->tNo = ++tNo; printf("\t\tmul eax, [%d]\n\t\tmov [%d], eax\n", p->c1->tNo, p->c2->tNo, p->tNo); return; }
    if(p->nType == nDIV) { cg(p->c1); cg(p->c2); p->tNo = ++tNo; printf("\t\tmov eax, [%d]\n\t\tcdq\n\t\tidiv ebx\n\t\tmov [%d], eax\n", p->c1->tNo, p->c2->tNo, p->tNo); return; }
    if(p->nType == nNUM) { p->tNo = ++tNo; printf("\t\tmov eax, %s\n\t\tmov [%d], eax\n", p->lexeme, p->tNo); return; }
    if(p->nType == nID) { get(p->lexeme); p->tNo = ++tNo; printf("\t\tmov eax, [%s]\n\t\tmov [%d], eax\n", p->lexeme, p->tNo); return; }
    return;
}
struct Node *makeNode(int nType, struct Node *c1, struct Node *c2){
    struct Node *p=(struct Node *)malloc(sizeof(struct Node));
    if(p==NULL) { printf("memory error"); exit(-1); }
    p->nType=nType;
    p->c1=c1;
    p->c2=c2;
    return p;
}
void cg_head(){
    printf("section .text\n");
    printf("\tglobal _start\n");
    printf("\textern _printf\n");
    printf("_start:\n");
}
void cg_tail(){
    printf("\tret\n");
    printf("section .data\n");
    printf("formatStr db \"%c dd 0Wn\"\n");
    for(int i=0; i<26; i++) if(ref[i]==1) printf("%c dd 0Wn", 'a'+i);
    for(int i=1; i<=tNo; i++) printf("\t%dd dd 0Wn", i);
}
int main(){
    if(!yparse()){
        cg_head(); cg(ast); cg_tail();
    }
    else printf("parsing fail\n");
    return 0;
}

%{
#include <string.h>
#include "yacc.tab.h"
%}

%%

"def"           { return DEF; }
"main"          { return MAIN; }
"print"         { return PRINT; }
"while"         { return WHILE; }
"[0-9]+"
"[a-z]"
["\t\n"]
[-;+*()]
.
%}

{ yyval.strVal=strdup(yytext); return NUM; }

{ yyval.strVal=strdup(yytext); return ID; }

{ }

{ return yytext[0]; }

{ return yytext[0]; }

%
```

#9: T 언어 (컴파일러, x86)

```
parser < input3.txt > input3.asm
nasm -f win32 input3.asm -o input3.o
ld -o pgm.exe input3.o -LC:\MinGW-w64\mingw32\i686-w64-mingw32\lib -lmsvcrt
pgm
13
```

input3.txt

```
def main()
    x=10;
    print x+3;
}
```

TAC

```
t1 = 10
x = t1
t2 = x
t3 = 3
t4 = t2 + t3
param t4
call print
```

input3.asm

```
section .text
    global _start
    extern _printf
_start:
    mov eax, 10
    mov [t1], eax
    mov eax, [t1]
    mov [x], eax
    mov eax, [x]
    mov [t2], eax
    mov eax, 3
    mov [t3], eax
    mov eax, [t2]
    add eax, [t3]
    mov [t4], eax
    mov eax, [t4]
    push eax
    push formatStr
    call _printf
    add esp, 8
    ret

section .data
formatStr db "%d",0
x dd 0
t1 dd 0
t2 dd 0
t3 dd 0
t4 dd 0
```

X=10 을 나타내는 코드

```
void cg(struct Node *p){
    if(p==NULL) return;
    if(p->nType==nSTMTS) {
        cg(p->c1);
        cg(p->c2);
        return;
    }
    if(p->nType==nASSIGN) {
        cg(p->c2);
        set(p->c1->lexeme,1);
        printf("Wtmov eax, [t%d]\n", p->c2->tNo);
        printf("Wtmov [%s], eax\n", p->c1->lexeme);
        return;
    }
    if(p->nType==nWHILE) {
        p->lNo1=++lNo;
        p->lNo2=++lNo;
        printf("L%d:\n", p->lNo1);
        cg(p->c1);
        printf("Wtmov eax, [t%d]\n", p->c1->tNo);
        printf("Wtcmp eax,0\n");
        printf("Wtje L%d\n", p->lNo2);
        cg(p->c2);
        printf("Wtjmp L%d\n", p->lNo1);
        printf("L%d:\n", p->lNo2);
        return;
    }
}
```

```
if(p->nType==nPRINT){
    cg(p->c1);
    printf("Wtmov eax, [t%d]\n", p->c1->tNo);
    printf("Wtpush eax\n");
    printf("Wtpush formatStr\n");
    printf("Wtcall _printf\n");
    printf("Wtadd esp, 8\n");
    return;
}
if(p->nType==nADD) {
    cg(p->c1);
    cg(p->c2);
    p->tNo=++tNo;
    printf("Wtmov eax, [t%d]\n", p->c1->tNo);
    printf("Wtadd eax, [t%d]\n", p->c2->tNo);
    printf("Wtmov [t%d], eax\n", p->tNo);
    return;
}
if(p->nType==nSUB) { ... }
if(p->nType==nMUL) { ... }
if(p->nType==nDIV) { ... }
if(p->nType==nNUM) {
    p->tNo=++tNo;
    printf("Wtmov eax, %s\n", p->lexeme);
    printf("Wtmov [t%d], eax\n", p->tNo);
    return;
}
if(p->nType==nID) {
    get(p->lexeme);
    p->tNo=++tNo;
    printf("Wtmov eax, [%s]\n", p->lexeme);
    printf("Wtmov [t%d], eax\n", p->tNo);
    return;
}
return;
```

#9: T 언어 (컴파일러, x86)

input4.txt

```
def main() {
    n=9;
    while(n){
        print n;
        n=n-1;
    }
}
```

TAC

```
t1 = 9
n = t1
L1:
t2 = n
iffalse t2 goto L2
t3 = n
param t3
call print
t4 = n
t5 = 1
t6 = t4 - t5
n = t6
goto L1
L2:
```

input4.asm

```
section .text
global _start
extern _printf
_start:
    mov eax, 9
    mov [t1], eax
    mov eax, [t1]
    mov [n], eax
L1:
    mov eax, [n]
    mov [t2], eax
    mov eax, [t2]
    cmp eax,0
    je L2
    mov eax, [n]
    mov [t3], eax
    mov eax, [t3]
    push eax
    push formatStr
    call _printf
    add esp, 8
    mov eax, [n]
    mov [t4], eax
    mov eax, 1
    mov [t5], eax
    mov eax, [t4]
    sub eax, [t5]
    mov [t6], eax
    mov eax, [t6]
    mov [n], eax
    jmp L1
L2:
    ret
```

```
section .data
formatStr db "%d",0
n dd 0
t1 dd 0
t2 dd 0
t3 dd 0
t4 dd 0
t5 dd 0
t6 dd 0
```

parser < input4.txt > input4.asm

```
nasm -f win32 input4.asm -o input4.o
ld -o pgm.exe input4.o -LC:\MinGW\w64\mingw32\lib -lmsvcrt
pgm
987654321
```

```
void cg(struct Node *p){
    if(p==NULL) return;
    if(p->nType==nSTMTS) {
        cg(p->c1);
        cg(p->c2);
        return;
    }
    if(p->nType==nASSIGN) {
        cg(p->c2);
        set(p->c1->lexeme,1);
        printf("Wtmov eax, [%d]\n", p->c2->tNo);
        printf("Wtmov [%s], eax\n", p->c1->lexeme);
        return;
    }
    if(p->nType==nWHILE) {
        p->lNo1=++lNo;
        p->lNo2=++lNo;
        printf("L%d:\n", p->lNo1);
        cg(p->c1);
        printf("Wtmov eax, [%d]\n", p->c1->tNo);
        printf("Wtcmp eax,0\n");
        printf("Wtje L%d\n", p->lNo2);
        cg(p->c2);
        printf("Wtjmp L%d\n", p->lNo1);
        printf("L%d:\n", p->lNo2);
        return;
    }
}
```

```
if(p->nType==nPRINT){
    cg(p->c1);
    printf("Wtmov eax, [%d]\n", p->c1->tNo);
    printf("Wtpush eax\n");
    printf("Wtpush formatStr\n");
    printf("Wtcall _printf\n");
    printf("Wtadd esp, 8\n");
    return;
}
if(p->nType==nADD) {
    cg(p->c1);
    cg(p->c2);
    p->tNo=++tNo;
    printf("Wtmov eax, [%d]\n", p->c1->tNo);
    printf("Wtadd eax, [%d]\n", p->c2->tNo);
    printf("Wtmov [%d], eax\n", p->tNo);
    return;
}
if(p->nType==nSUB) { ... }
if(p->nType==nMUL) { ... }
if(p->nType==nDIV) { ... }
if(p->nType==nNUM) {
    p->tNo=++tNo;
    printf("Wtmov eax, %s\n", p->lexeme);
    printf("Wtmov [%d], eax\n", p->tNo);
    return;
}
if(p->nType==nID) {
    get(p->lexeme);
    p->tNo=++tNo;
    printf("Wtmov eax, [%s]\n", p->lexeme);
    printf("Wtmov [%d], eax\n", p->tNo);
    return;
}
return;
```

References

- ▣ 박두순. (2016). (내공 있는 프로그래머로 길러주는)컴파일러의 이해. 한빛아카데미.
- ▣ 창병모. (2021). 프로그래밍 언어론 : 원리와 실제. 인피니티북스.
- ▣ Aho, A., Lam, M., Sethi, R., Ullman, J. (2006). Compilers: Principles, Techniques, and Tools (2nd Ed.). Addison Wesley.
- ▣ Nystrom, R. Crafting interpreter. <https://craftinginterpreters.com/>
- ▣ Sebesta, R. (2012). Concepts of Programming Languages (10th. ed.). Pearson.
- ▣ Thain, D. (2023). Introduction to Compilers and Language Design.
- ▣ Paxson, V. (1995). Flex, version 2.5.
- ▣ Donnelly, C., Stallman, R. (2008). Bison.
- ▣ LexAndYacc.pdf (epaperpress.com)
- ▣ Tom Niemann, LEX & YACC. <http://epaperpress.com/lexandyacc>