

Lex & Yacc (flex, bison)

국민대학교 소프트웨어학부
강 승 식

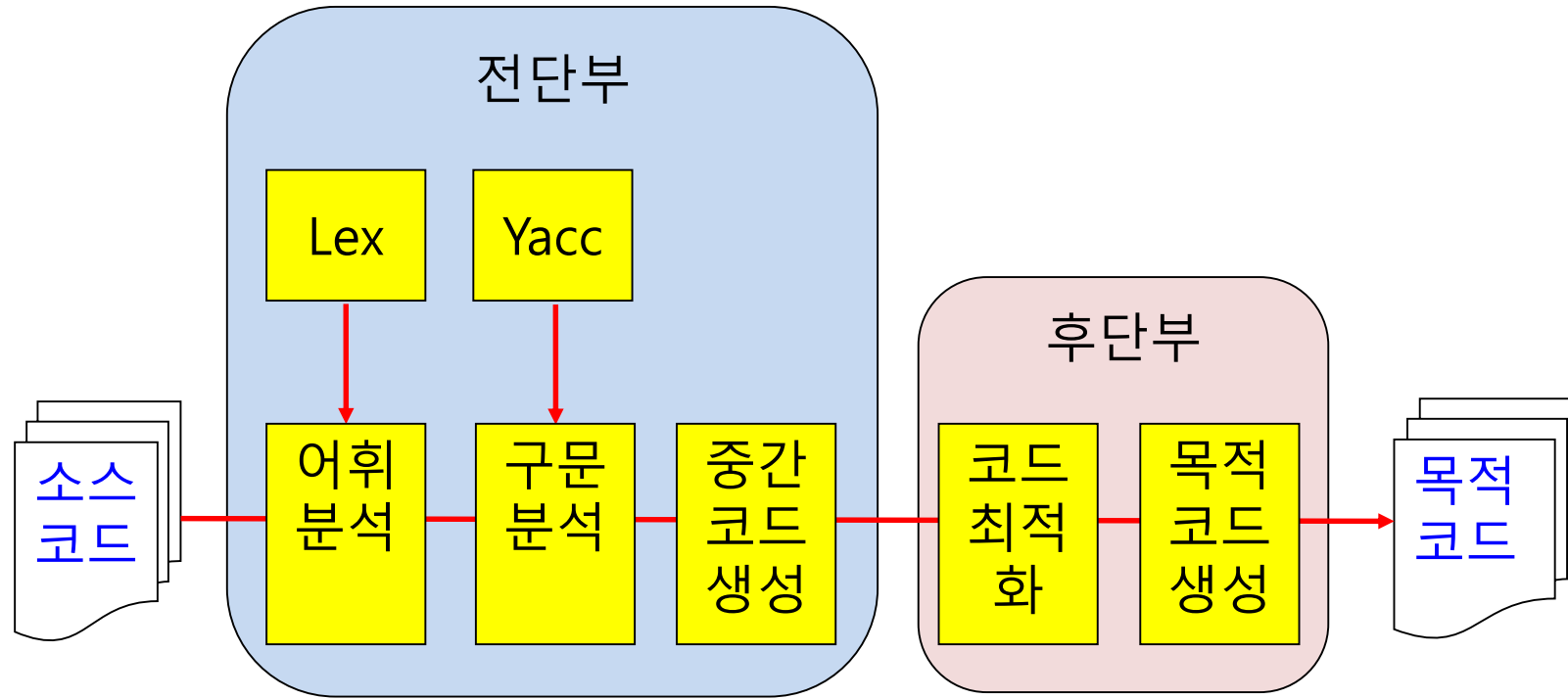


그림 3.1 컴파일러 구조

<참고> Linux 라이브러리: /usr/lib64

```
[root@nlp /]# ls /usr/lib64/lib*.a
/usr/lib64/libBrokenLocale.a  /usr/lib64/libfl.a          /usr/lib64/libm.a
/usr/lib64/libacl.a           /usr/lib64/libfl_pic.a      /usr/lib64/libmcheck.a
/usr/lib64/libanl.a           /usr/lib64/libg.a           /usr/lib64/libname-server-2.a
/usr/lib64/libattr.a          /usr/lib64/libgdbm.a        /usr/lib64/libnsl.a
/usr/lib64/libbfd.a           /usr/lib64/libgnutls-extra.a /usr/lib64/libopcodes.a
/usr/lib64/libbsd-compat.a    /usr/lib64/libgnutls-openssl.a /usr/lib64/libpci.a
/usr/lib64/libbsd.a           /usr/lib64/libgnutls.a      /usr/lib64/libpthread.a
/usr/lib64/libc.a             /usr/lib64/libgnutlsxx.a    /usr/lib64/libpthread_nonshared.a
/usr/lib64/libc_nonshared.a   /usr/lib64/libhesiod.a      /usr/lib64/libresolv.a
/usr/lib64/libc_stubs.a       /usr/lib64/libiberty.a      /usr/lib64/librpcsvc.a
/usr/lib64/libcom_err.a       /usr/lib64/libieee.a        /usr/lib64/librt.a
/usr/lib64/libcrmfs.a         /usr/lib64/libl.a           /usr/lib64/librtkaios.a
/usr/lib64/libcrypt.a         /usr/lib64/liblber.a        /usr/lib64/libutil.a
/usr/lib64/libdl.a            /usr/lib64/libldap.a        /usr/lib64/libuuid.a
/usr/lib64/libexpat.a         /usr/lib64/libldap_r.a      /usr/lib64/libxslt.a
/usr/lib64/libexslt.a         /usr/lib64/liblockdev.a
[root@nlp /]#
```

<참고> Static Library 생성 방법

```
$ gcc -c a.c
```

```
$ gcc -c b.c
```

```
$ gcc -c c.c
```

```
$ ar rcv libtest.a *.o
```

```
$ gcc -c test.c
```

```
$ gcc -o test.exe test.o -ltest
```

Shared Object Library 생성 및 사용법은?
(윈도에서는 DLL)

실습 환경 – 윈도우에서 Cygwin

<http://www.cygwin.com>

Cygwin

- Install Cygwin
- Update Cygwin
- Search Packages
- Licensing Terms

Cygwin/X

Community

- Reporting Problems
- Mailing Lists
- Newsgroups
- IRC channels
- Gold Stars
- Mirror Sites
- Donations

Documentation

- FAQ
- User's Guide
- API Reference
- Acronyms

Contributing

- Snapshots
- Source in Git
- Cygwin Packages

Related Sites

Cygwin

Get that [Linux](#) feeling - on Windows

Installing and Updating Cygwin Packages

Installing and Updating Cygwin for 32-bit versions of Windows

Run [setup-x86.exe](#) any time you want to update or install a Cygwin package for 32-bit windows. The [signature](#) for [setup-x86.exe](#) can be used to verify the validity of this binary using [this](#) public key.

Installing and Updating Cygwin for 64-bit versions of Windows

Run [setup-x86_64.exe](#) any time you want to update or install a Cygwin package for 64-bit windows. The [signature](#) for [setup-x86_64.exe](#) can be used to verify the validity of this binary using [this](#) public key.

General installation notes

When installing packages for the first time, `setup*.exe` does not install every package. Only the **minimal base packages** from the Cygwin distribution are installed by default, which takes up about 100 MB.

- All Default
- Admin Install
- Archive Install
- Audio Default
- Base Install
- Database Default
- Debug Default
- Devel Install
- Doc Default
- Editors Install
- Games Default
- GNOME Install
- Graphics Default
- Interpreters Install
- KDE Default
- Libs Install
- Mail Default
- Math Default
- Misc Default
- Net Default
- Perl Install
- Publishing Install
- Python Install
- Ruby Install
- Science Default
- Shells Install
- System Install
- Tcl Default
- Text Install
- Utils Default
- Video Default
- Web Default
- X11 Default

Lex 입력 작성 예

정의부
%%
규칙부
%%
사용자 서브루틴

```
// cat.l
%%
. | \n    ECHO;
%%
main() {
    yylex();
}
```

```
// cat2.l
%{
    #include <stdio.h>
%}
%%
. | \n { printf("%s\n", yytext); }
%%
main() {
    yylex();
}
```

컴파일 및 실행

- Unix 환경에서

```
$ lex cat.l  
$ gcc lex.yy.c -ll  
$ ./a.out
```

- Linux 환경에서

```
$ flex cat.l  
$ gcc lex.yy.c -lfl  
$ ./a.out
```

```
// “-lfl” 옵션 작동 안할 때 yywrap() 추가  
int yywrap( ) { return 1; }
```

Lex 메타 문자

메타 문자	의미
.	줄 바꿈("\n")을 제외한 모든 문자와 일치된다.
[]	대괄호 안의 문자들 중 하나에 매치한다. [a-z]는 알파벳 소문자 a~z중 하나가 매치될 수 있음을 의미한다.
*	0번 이상 반복을 의미한다. ab*c는 ac, abc, abbc, ab...bc를 나타낸다.
+	1번 이상 반복을 의미한다. ab+c는 abc, abbc, ab...bc를 나타낸다.
?	? 앞 문자를 선택적으로 일치한다. ab?c는 ac 또는 abc를 나타낸다.
{ }	중괄호는 앞에 있는 패턴의 반복 횟수를 의미한다. a{3}은 aaa를 나타낸다. b{1:3}은 b, bb, bbb를 나타낸다. 정의절에서 정의된 이름을 규칙절에서 치환식으로 사용하고자 할 때도 사용한다.
\ 또는 \w	c언어의 이스케이프 문자를 표현할 때 사용한다. '*' 문자를 *로 기술하면 메타문자로 인식된다. 문자 자체를 일치하도록 하려면 * 와 같이 기술한다.
()	정규식을 하나의 단위로 간주할 때 사용한다. a(b c)는 ab 또는 ac를 나타낸다.
	OR 표현을 나타낸다. ab ac는 ab 또는 ac를 의미한다.
" "	따옴표 안에 있는 내용은 문자 그대로를 의미한다. "*"는 *와 같다.
/	접미 문맥을 명시할 때 사용한다. ab/cd는 ab 다음에 cd가 연속되어 나타나는 ab만을 인식하겠다는 의미이다.
^	라인의 첫문자 위치를 명시할 때 사용한다. ^ab는 "ab"가 그 줄의 첫문자로 시작되는 경우에만 인식된다. "aabcd"는 ab가 첫 문자로 시작되지 않으므로 인식되지 않는다. '^'이 대괄호 안에 사용될 때는 not 연산자로서 여집합을 의미한다. [^a-z]는 알파벳 소문자 a~z를 제외한 모든 문자를 나타낸다.
\$	라인의 끝 위치를 명시할 때 사용한다. ab\$는 문장 "cdab\n"과 같이 라인끝에 ab가 있는 경우에만 ab가 인식된다. "cdabb"는 ab가 라인 끝 위치가 아니므로 인식되지 않는다.

정의부 사용 예

// 정의부 사용 예

```
DIGIT    [0-9]+
EOL      \n
%%
{DIGIT}{EOL} { printf("number\n"); }
%%
main()
{
    yylex();
}
```

// digit.l

```
%%
[0-9]+ { printf("int\n"); }
[0-9]*\.[0-9]+ { printf("float\n"); }
(-|\+)[0-9]+ { printf("signed int\n"); }
(-|\+)[0-9]*\.[0-9]+ { printf("signed float\n"); }
%%
main()
{
    yylex();
}
```

```
DIGIT    [0-9]+
%%
{DIGIT}+ { printf("int\n"); }
{DIGIT}*\.[0-9]+ { printf("float\n"); }
(-|\+){DIGIT}+ { printf("signed int\n"); }
(-|\+){DIGIT}*\.[0-9]+ { printf("signed float\n"); }
%%
main()
{
    yylex();
}
```

test1.l

```
%{
    #include <stdio.h>
}%
%%
a*b      { printf("a...b\n"); }
b*a      { printf("b...a\n"); }
c*       { printf("c...\n"); }
ab*      { printf("ab...\n"); }
.*       { printf("none of above\n"); }
%%
int yywrap()
{
    return 0;
}

main()
{
    yylex();
}
```

variable.l : 명칭 인식

```
%{
    /* C 언어의 변수 인식 */
    #include <stdio.h>
}%
%%
LETTER [a-zA-Z]
DIGIT [0-9]
UNDERBAR _
%%
"이 부분을 완성하십시오." { printf("\t%s -> variable\n", yytext); }
.; //skip
%%
main()
{
    yylex();
}
```

test2.l

```
%{
    #include <stdio.h>

%}
DIGIT      [0-9]
LETTER     [A-Za-z]
%%
[0-9]+     { printf("정수\n"); }
[+|-]{DIGIT}+ { printf("부호있는 정수\n"); }
[0-9]+W.[0-9]* { printf("실수\n"); }
int        { printf("INT\n"); }
char       { printf("CHAR\n"); }
float      { printf("FLOAT\n"); }
[a-z]+     { printf("소문자 스트링\n"); }
[A-Z]+     { printf("대문자 스트링\n"); }
{LETTER}+  { printf("대소문자 혼합 스트링\n"); }
[{LETTER} | {DIGIT}] + { printf("영문-숫자 혼합 스트링\n"); }
.*        { printf("기타 나머지 스트링\n"); }
%%
int yywrap() { }
main() { yylex(); }
```

test3.l : printf → return 문

```
%{
    #include <stdio.h>
    enum { UNUSED, INT, SINT, FLOAT, INT_TYPE, CHAR_TYPE, FLOAT_TYPE, LOW_STR,
           UPP_STR, LOWUPP_STR, ALPHANUM_STR, ETC };

%}
DIGIT      [0-9]
LETTER     [A-Za-z]
%%
[0-9]+     { return INT; }
[+|-]{DIGIT}+ { return SINT; }
[0-9]+W.[0-9]* { return FLOAT; }
int        { return INT_TYPE; }
char       { return CHAR_TYPE; }
float      { return FLOAT_TYPE; }
[a-z]+     { return LOW_STR; }
[A-Z]+     { return UPP_STR; }
{LETTER}+  { return LOW_UPP_STR; }
[{LETTER} | {DIGIT}] + { return ALPHANUM_STR; }
.*        { return ETC; }
%%
int yywrap() { }
main()
{
    enum n;

    n = yylex();
    printf("Token number = %d\n", i);
}
```

test4.l : word count -- yyin, yyout, yyleng

```
%{
    unsigned char_count=0, word_count=0, line_count=0;
}%
WORD [^ \t\n]+
EOL \n
%%
{WORD} { word_count++; char_count += yyleng; }
{EOL}  { line_count++; char_count++; }
.      char_count++;
%%
void main(int argc, char *argv[])
{
    if (argc > 1) {
        FILE *file;
        file = fopen(argv[1], "r");
        if (!file) {
            fprintf(stderr, "could not open %s\n", argv[1]);
            exit(1);
        }
        yyin = file;
    }
    yylex();
    fprintf(yyout, "%d %d %d\n", line_count, word_count, char_count);
    fclose(yyin);
}
```

규칙부의 정규식 우선순위

int.l

```
%{  
    /* 렉스 패턴 매치 우선순위 예제 */  
%}  
%%  
in { printf("IN\n"); }           // R1  
in. { printf("IN+CHAR\n"); }     // R2  
int { printf("INT\n"); }         // R3  
. { printf("CHAR: %s\n", yytext); } // R4  
%%  
main()  
{  
    yylex();  
}
```

- 1) 입력 스트링 ing 은 "in"과 "in."와 "int" 중 어떤 정규식이 매치되는가?
<참고> ing 는 아래와 같이 다양하게 인식될 수 있다.

ing: R2
in + g: R1+R4
i + n + g: R4+R4+R4

- 2) 입력 스트링 int 는 "in"과 "in."와 "int"중 무엇이 매치되는가?
<참고> int 는 아래와 같이 다양하게 인식될 수 있다.

int: R2
int: R3
in + t: R1+R4
i + n + t: R4+R4+R4

Lex 입력 연습: 빈곳 완성(명칭, 정수)

variable.l

```
%{
    /* C 언어의 변수 인식 */
    #include <stdio.h>
}%

LETTER [a-zA-Z]
DIGIT [0-9]
UNDERBAR _

%%
"이 부분을 완성하십시오." { printf("%%t%%s -> variable%%n", yytext); }
. ; //skip
%%
main()
{
    yylex();
}
```

integer.l

```
%{
    /* C 언어의 정수 인식 */
    #include <stdio.h>
}%

DIGIT [0-9]
OCTAL [0-7]
HEXA [0-9a-fA-F]

%%
"이 부분을 완성하십시오." { printf("%%t%%s -> 8진수 정수%%n", yytext); }
"이 부분을 완성하십시오." { printf("%%t%%s -> 10진수 정수%%n", yytext); }
"이 부분을 완성하십시오." { printf("%%t%%s -> 16진수 정수%%n", yytext); }
%%
main()
{
    yylex();
}
```

Lex 입력 연습: 빈곳 완성(실수, 스트링)

real_number.l

```
%{
    /* C 언어의 실수 인식 */
}%
DIGIT [0-9]
EXPONENT [eE]
SIGN [₩+-]
%%
"이 부분을 완성하십시오." { printf("₩t₩s -> 실수 상수₩n", yytext); }
. ; //skip
%%
main()
{
    yylex();
}
```

string_constant.l

```
%{
    /* 스트링 상수 인식 */
    #include <stdio.h>
}%
LETTER [a-zA-Z]
QUOTE ₩"
ESCAPE_QUOTE ₩₩₩
%%
"이 부분을 완성하십시오." { printf("₩t₩s -> 스트링 상수₩n", yytext); }
. ; //skip
%%
main()
{
    yylex();
}
```

Lex 입력 연습: 코멘트, C언어의 키워드

comment.l

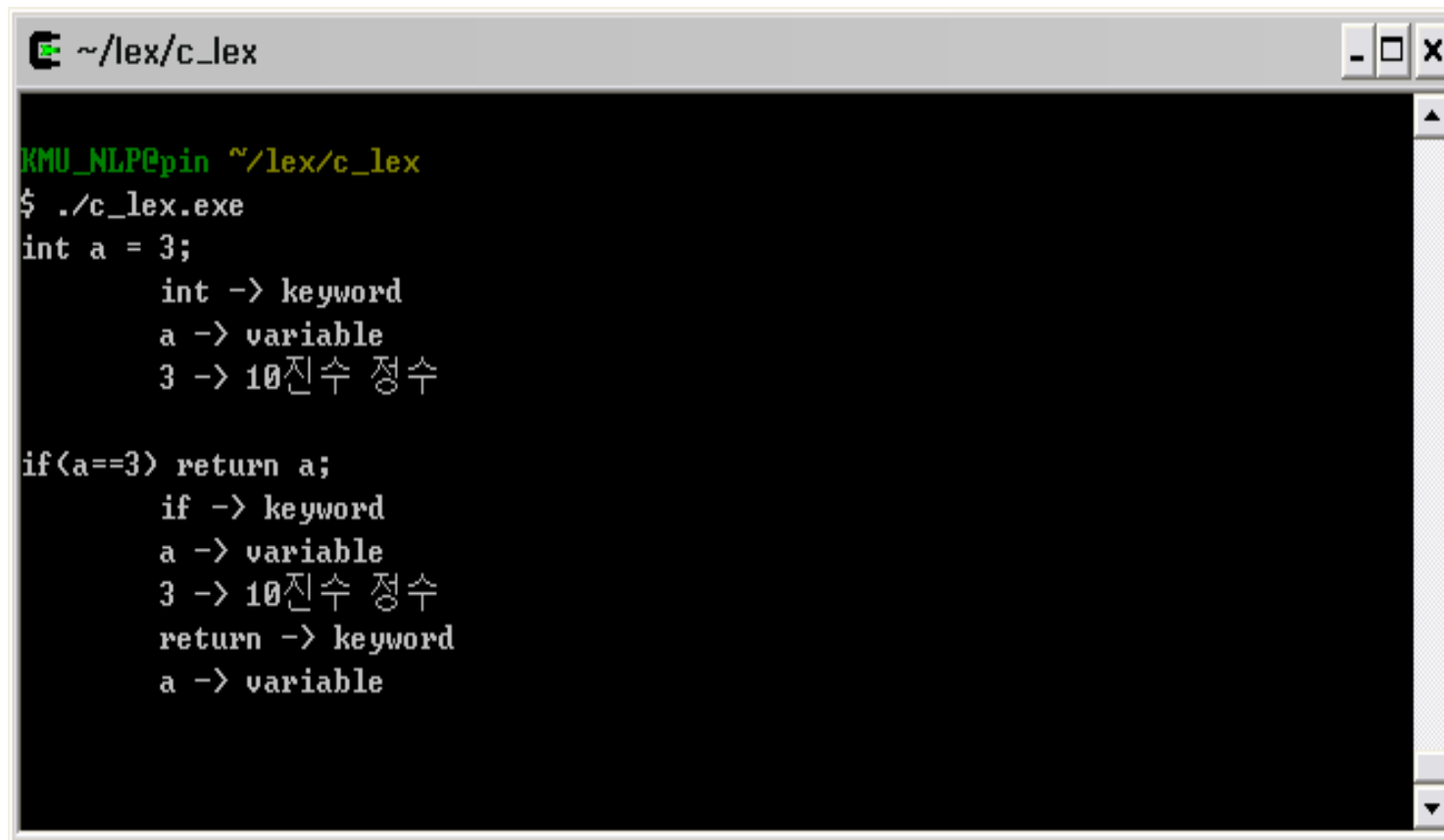
```
%{
    /* 주석 인식 */
    #include <stdio.h>
}%
C [^\\n\\*]
STAR [\\*]
BEGIN1 "//"
BEGIN2 "/\\*"
END "\\*/"
%%
"이 부분을 완성하십시오." {printf("\\t%s -> //주석 \\n", yytext);}
"이 부분을 완성하십시오." {printf("\\t%s -> /**주석 \\n", yytext);}
. ; //skip
%%
main()
{
    yylex();
}
```

keyword.l

```
%{
    /* keyword */
}%
%%
"이 부분을 완성하십시오." { printf("\\t%s -> keyword\\n", yytext); }
. ; //skip
%%
main()
{
    yylex();
}
```


Lex 입력 연습: C언어의 문장

- 아래 예와 같이 C언어 문장 입력에 대한 Lex 입력을 작성하시오.

A terminal window titled '~ / lex / c_lex' with standard window controls. The prompt is 'KMU_NLP@pin ~/lex/c_lex'. The user enters '\$./c_lex.exe'. The terminal shows the Lex input for two C statements. The first statement is 'int a = 3;', with token mappings: 'int' to 'keyword', 'a' to 'variable', and '3' to '10진수 정수'. The second statement is 'if(a==3) return a;', with token mappings: 'if' to 'keyword', 'a' to 'variable', '3' to '10진수 정수', 'return' to 'keyword', and 'a' to 'variable'.

```
~/lex/c_lex
KMU_NLP@pin ~/lex/c_lex
$ ./c_lex.exe
int a = 3;
    int -> keyword
    a -> variable
    3 -> 10진수 정수

if(a==3) return a;
    if -> keyword
    a -> variable
    3 -> 10진수 정수
    return -> keyword
    a -> variable
```

함수 yywrap()

- Lex는 입력 파일의 끝인 EOF문자를 만났을 때 함수 yywrap()을 호출
- yywrap()은 입력의 종료를 알리는 값으로 1을 반환한다.
 - 반환값이 0이면 Lex는 계속해서 새로운 입력을 받아들임.
- 예제) yywrap() 함수를 재정의하여 여러 개의 파일을 모두 읽어서 문자, 단어, 라인의 빈도수를 계산하는 Lex 입력 파일

연습 1. Lex 입력 파일 작성

- 아래 수식에 대해 각 토큰을 인식하는 Lex 입력 파일을 작성하시오.

**$-(15.7 + -2^3) * 3.14 - (12.34 / \text{sqrt}(2) +$
 $\text{abs}(-12) \% -7) + \sin(x+y)) + \cos(\text{var1}-\text{var2})$**

- 위 식에서 '^'는 지수승, '%'는 modula(정수 나눗셈의 나머지) 연산이다.
- 출력 방식: 각 토큰에 대해 우측 예제와 같이 한 라인에 한 개씩 <인식된
스트링, 토큰명> 출력

<참고> INT, REAL, ... VAR를 열거형으로 정의하여 정수, 실수, 명칭(변수)에 대해서는 <token number, token value>, 기타 토큰들에 대해서는 <token number, 0>을 출력한다.

enum { ERROR, INT, REAL, PLUS, MINUS, ..., VAR };

```
정수: INT
실수: REAL
'+' : PLUS
'-' : MINUS
'*' : MUL
'/' : DIV
'%' : REM
'^' : EXP
'(' : LPAREN
')' : RPAREN
sqrt: Sqrt
sin: SIN
cos: COS
abs: ABS
x, y, var1, var2: VAR
```

연습 2. C 소스의 토큰 인식

- 우측 C 소스에 대해 각 토큰들을 인식하는 프로그램을 Lex를 이용하여 작성하시오.
- 인식된 토큰은 <token number, token value> 형태로 출력하고, token number는 열거형으로 정의
- Comment는 무시

```
/* Bubble sort program */
int i, j, n, temp; // temporary variables
int sum=0, arr[100];

for (i=0; i < n-1; i++) {
    for (j=i+1; j < n; j++) {
        if (arr[i] > arr[j]) {
            temp = arr[i];
            arr[i] = arr[j];
            arr[j] = temp;
        }
    }
}

while (--n >= 0) sum += arr[n];
```