Lex & Flex

COMP321 컴파일러

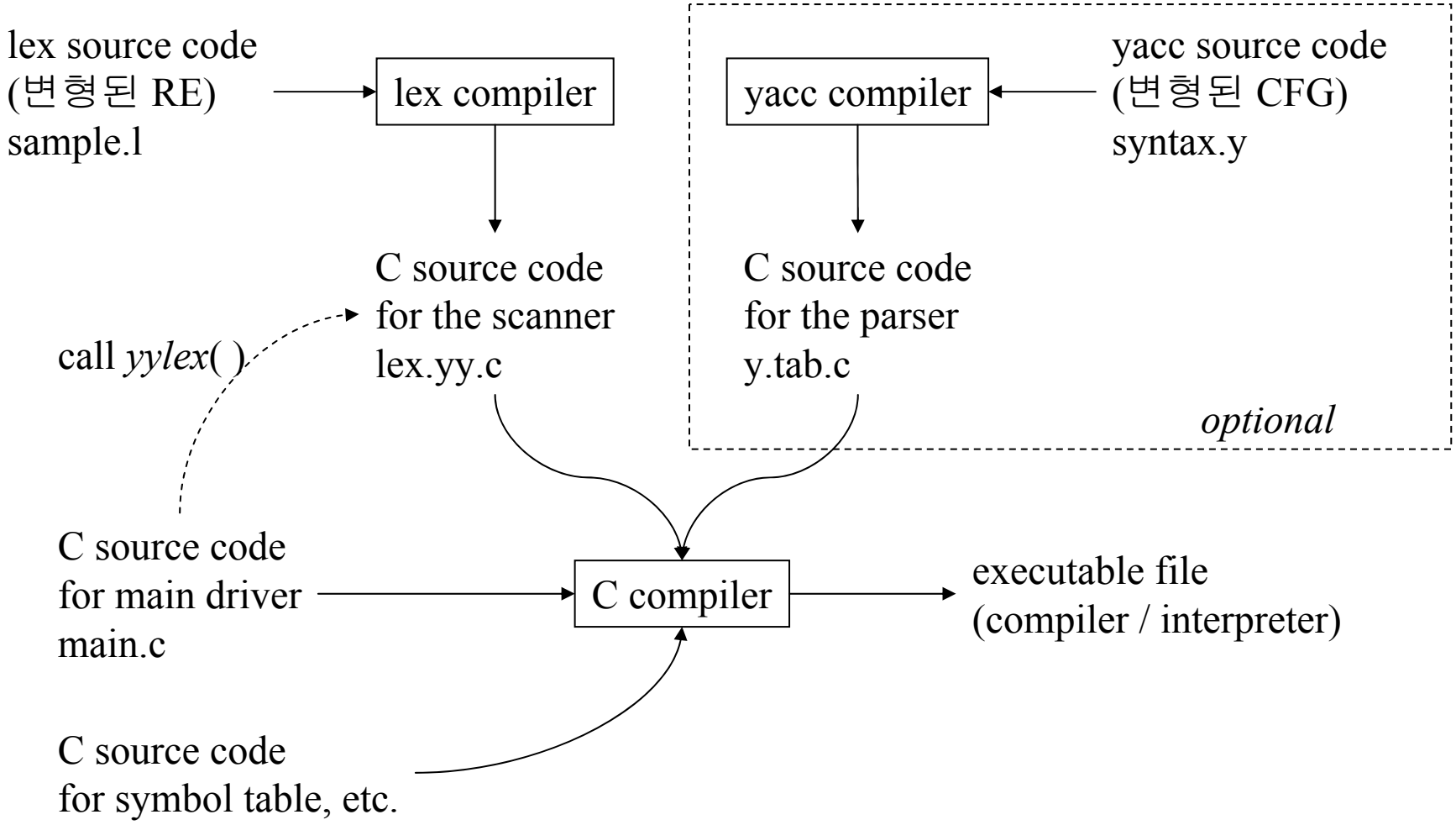
2007년 가을학기

경북대학교 전자전기컴퓨터학부

Lex : A Lexical Analyzer Generator

- 1975년 M. E. Lesk & Schmidt @ Bell Lab.
  - M. E. Lesk and E. Schmidt,
    "LEX – Lexical Analyzer Generator",
    Unix Research System Programmer's Manual,
    Tenth Edition, Volume 2.
- **A lexical analyzer generator**
  - goal : automated scanner generation
  - input : lex program
  - output : **C source code for the lexical analyzer**
- **Flex** : a fast scanner generator
  - GNU re-implementation of Lex (+ extra features)

Global View


lex source code
(변형된 RE)
sample.l
lex compiler
yacc compiler
yacc source code
(변형된 CFG)
syntax.y
C source code
for the scanner
lex.yy.c
C source code
for the parser
y.tab.c
optional
call yylex( )
C source code
for main driver
main.c
C compiler
executable file
(compiler / interpreter)
C source code
for symbol table, etc.

Lex Source Code

- 전체 구성

  definitions : 미리 정의할 것들

  %%

  rules : 변형된 RE + 대응되는 action

  %%

  user subroutines : 추가로 필요한 C function 정의

- %% : 각 부분의 구분자
  - 두 번째 %% 이후는 optional (생략 가능)
  - 첫 번째 %%는 생략 불가능
- 각 부분은 순서를 지켜야 함

Rules Part

- Lex source code 의 핵심
  - **(regular expression) (space) (action)** 형태의 반복
  - RE 에 match 되는 문자열을 만나면, action 수행
  - action 은 **C source code**

- example

```
%%
integer    printf("found keyword INT");
```

Regular Expression

- Lex RE : 변형된 regular expression
  - 왜 변형? keyboard 입력이 쉽게
- 텍스트 문자와 연산자 문자들로 구성
  - 텍스트 문자 : matching 되는 글자를 의미
  - 연산자 문자 : 반복 또는 선택 등을 표시

    " \ [ ] ^ - ? . * + ( ) $ / { } % < >

- " (quote) : " 사이에 있는 모든 문자는 텍스트 문자로
  - example: ">>"
- \ (backslash) : 한 개의 문자를 텍스트 문자로
  - example: \t \\ \"

Regular Expression

- **[ ]** : 문자들의 클래스.
  - 예: [abc] → a or b or c
  - in [ ] operator,
    - – (dash) : range operator
      - 예: [0-9] → 0, 1, 2, …, 8, 9
    - ^ (hat) : complement operator
      - 예: [^a-zA-Z] → alphabet을 제외한 글자
    - \ (backslash) : C에서와 같은 의미
- **?** : 바로 앞 element가 선택적(optional)
  - 예: ab?c → abc or ac

Regular Expression

- **.** (period) : new line 을 제외한 모든 문자
- * : 0번 이상의 반복
- + : 1번 이상의 반복
  - 예: [_A-Za-z][_A-Za-z0-9]* → C identifier
- | : or
  - 예: a(b|c)d → abd or acd
- ^ : start of line
- **$** : end of line
  - 예: ^abc → line 처음에 오는 abc만
  - 예: ^[\t]*$ → blank line

Regular Expression

- **{ numbers }** : repetition
  - 예: a{1,5} → a, aa, aaa, aaaa, aaaaa
- **{ name }** : definition expansion
  - 예: digit [0-9]
    %%
    { digit } → [0-9] 와 동일 !

Actions

- regular expression matching 시에 수행할 행동
  - C source code 형태 (그대로 수행됨)
- *special action* **;** : no action
- *special action* **ECHO** : print the matching string
- { } 사용 가능 : 여러 문장 수행 시
- default action : no-matching 시, ECHO 수행
- special variable ***yytext***
  - **matching string**을 저장하는 char array
  - 예: [0-9]+ printf("%s", ***yytext***);

An Example

- in sample0.l

```
%%
[ \t]+$    ;
[ \t]+     printf(" ");
```

- remove all blanks or tabs at the ends of lines
- replace multiple blanks / tabs to a single space

- 간단한 text 정리 프로그램 !

- compile it ! (in Unix / Linux)

```
$ lex sample0.l
    (you got lex.yy.c)
$ cc lex.yy.c –ll
    (you got a.exe)
$ cat input.txt
Hello,     my world.
Smile     Again.
$ a.exe < input.txt
Hello, my world.
Smile Again.
```

Another Example

- in sample1.l

```
%%
[a-z]+          printf("var = %s\n", yytext);
[0-9]+          { long val = strtol(yytext, NULL, 10);
                  printf("int = %d\n", val);
                }
\n|.            ; /* ignore all others */
```

Tie-Break Rule

- 2개 이상의 RE에서 matching 되면?
  - 가장 길게 matching 되는 것 선택
    - 예: rule에 =, <, <= 모두 정의
      <= 는 <= 로 인식.
  - 같은 길이이면, 먼저 나오는 것 선택
    - 예: rule에서,
      for /* token for */
      [a-zA-Z]+ /* identifier */
    - token으로 인식

Definitions Part

- definition part format

```
%{
    /* 이 부분은 lex.yy.c 속으로 그대로 복사 */
%}
name1       expression1
name2       expression2
…
```

- %{ 와 %} 사이 : 여러 번 사용 가능
  - #include, global variable, function 등등을 정의
  - lex.yy.c의 앞부분에 그대로 복사.
- name space transitions : macro 기능 (#define 과 유사)
  - digit   [0-9]

User Subroutines Part

- 2번째 %% 이후는
  - Lex에 의한 어떤 처리도 없이 그대로 **lex.yy.c의 제일 뒤에 복사**
  - 주로 symbol table handling 등의 function 구현

- main( ) 을 정의해도 되고, 안 해도 됨
  - 정의하지 않으면, Lex 의 **default main( )** 사용

Lex Variables and Functions

- Lex 내부에서 제공하는 변수와 함수
  - rule part와 subroutine part에서 사용 가능
- char yytext[]; matching 된 string 전체 저장
- int yyleng; matching된 string 길이
  - char c = yytext[yyleng – 1]; // 마지막 글자
- input( ) read the next character from input stream
- output(c) output the character c to the output stream
- unput(c) push back the character c to the input stream
  - back-trace 가 필요한 경우를 위해…
- yywrap( ) 여러 file 처리 시, 특별한 용도로 사용

Main and yylex( ) function

- main( ) 함수 사용 시, **yylex( ) 를 불러야 기능 수행**
- simple case:
  - EOF 까지 계속 RE matching 수행
  - EOF 시에, yywrap( )을 call
    - return 값이 1 (true) 이면 return to main
    - otherwise, continue (yywrap에서 stdin을 새로 설정해야 함)
- single token case:
  - action 부분에 return 을 쓰면, return to main
  - token 하나씩을 받을 때, 유용

Lex Compile 방법

- in Unix / Linux / Cygwin,
  - lex / flex 명령 수행
  - $ lex sample.l
  - $ flex sample.l
    - lex.yy.c 를 생성
  - $ gcc lex.yy.c –ll (Lex 용)
  - $ gcc lex.yy.c –lfl (Flex 용)
    - -ll / -lfl : link Lex / Flex library
    - -ll / -lfl 안에 yytext, yyleng, main( ) 등 정의
- in MS Visual Studio,
  - special version 필요 → see class homepage

An Example : word counter

- UNIX wc 명령과 동등

```
%{
#include <stdio.h>
int nchar, nword, nline;
%}
%%
\n              ++nchar, ++nline;
[^ \t\n]+       { ++nword, nchar +=yyleng;
                  printf("word: %s\n", yytext);
                }
.               ++nchar;
%%
int main(void) {
        yylex();
        printf("number of line: %d\n", nline);
        printf("number of word: %d\n", nword);
        printf("number of character: %d\n", nchar);
}
```

Another Example

```
%{
#define IDEN 1
#define NUM 2
long yyvalue;
%}
%%
[a-z]+  { return IDEN; }
[0-9]+  { yyvalue = strtol(yytext, NULL, 10); return NUM; }
\n|.    ; /* ignore all others */
%%
int yywrap(void) {
        printf("yywrap() called.\n");
        return 1;
}

int main(void) {
        int code;
        while ((code = yylex()) != 0) {
                switch (code) {
                case IDEN:      printf("identifier: %s\n", yytext); break;
                case NUM:       printf("number: %d\n", yyvalue); break;
                }
        }
}
```