

Lab01. Lexical Analysis

조교 : 박성환

starjara@pusan.ac.kr



Introduction

- 컴파일러 실습 목표

- PL/0'이라는 언어를 위한 컴파일러 작성
 - 기존의 PL/0라는 언어에 함수를 추가한 언어
 - 해당 교제만을 위한 언어
 - PL/0언어는 기존 컴파일러 교육을 위해 널리 사용되었음
 - 기존 PL/0컴파일러는 파스칼로 작성되었으나 해당 실습에서는 C로 작성하여 본다
- 일반적인 기계어 코드가 아닌 스택을 가진 가상 머신에서 실행할 수 있는 후위 표기법 형태의 코드 생성
- 일반적인 기계어 목적 코드를 생성하는 것과 최적화는 고려하지 않음

Environments

- 실습환경
 - Ubuntu 18.04
 - 개인 노트북 사용 가능
 - VMware상에 설치하여 진행
- Pre-requirement
 - yacc (bison)
 - lex (flex)
 - gcc
- `sudo apt-get install bison flex`

Lexical Analysis

- 해당 실습 목표

- 컴파일과정의 첫 단계인 lexical analysis (낱말 분석)단계를 직접 구현
- 해당 실습을 통해 lexical parser를 구현하고 출력 결과를 직접 확인한다

Lex

- 어휘 분석 생성기를 만들기 위한 언어
- 다음과 같은 구조로 이루어져있다

{

선언부

}

{

규칙부

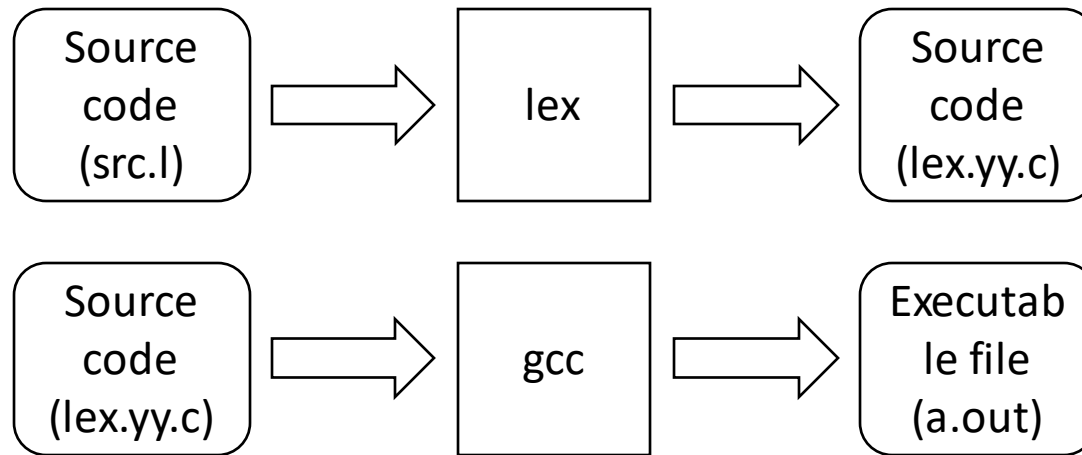
}

사용자 프로그램

- 정규표현식 지원

Lex

- 다음 과정을 거쳐 실행 파일 생성



- `$ lex src.l`
- `$ gcc lex.yy.c -lfl`
 - (해당 명령어에서 에러 발생시 `$ gcc lex.yy.c -ll` 로 시도)

부동소수점 상수 읽기

float.l

```
%{  
%}  
  
digit [0-9]  
integer {digit}+  
floating {integer}\.{integer}  
point '\.'  
  
%%  
{integer} { printf("Integer: %d\n", atoi(yytext)); }  
{floating} { printf("Float: %f\n", atof(yytext)); }  
%%  
  
int main (int argc, char *argv[])  
{  
    yylex();  
    return 0;  
}
```

Result

```
java@java-System-Product-Name:~/ETC/CompilerLab/Lab01.LexicalAnalysis/EX$ ./a.out  
3.765  
Float : 3.765000  
  
2  
Integer : 2  
  
9125  
Integer : 9125  
  
92.715  
Float : 92.715000
```

문자 읽기

wordCount.l

```
%{
unsigned charCount=0, wordCount=0, lineCount=1;
}%

word [^ \t\n]+
eol \n
%%

{word} {wordCount++; charCount+=yyleng;
printf("word %s\n", yytext);}

{eol} {charCount++; lineCount++;}

. charCount ++;
%%

main ()
{
yylex ();
printf("number of character : %d\n", charCount);
printf("number of word : %d\n", wordCount);
printf("number of line : %d\n", lineCount);
}
```

Input file

The token descriptions that lex uses are known as regular expressions, extended

versions of the familiar patterns used by the grep and egrep commands. A lex lexer is

almost always faster than a lexer that you might write in C by hand.

```
java@java-System-Product-Name:~/ETC/PL0 yacc$ ./a.out < test.l
word The
word token
word descriptions
word that
word lex
word uses
word are
word known
word as
word regular
word expressions,
word extended
word versions
word of
word the
word familiar
word patterns
word used
word by
word the
word grep
word and
word egrep
word commands.
word A
word lex
word lexer
word is
word almost
word always
word faster
word than
word a
word lexer
word that
word you
word might
word write
word in
word C
word by
word hand.
number of character : 236
number of word : 42
number of line : 4
```


주석 읽기

comment.l

```
%{  
%}  
  
commentSingle "//"  
commentMultBegin "/*"  
commentMultEnd "*/"  
  
%x CMNT  
%%  
{commentSingle}{. *} { printf("Single line comment\n"); }  
{commentMultBegin} BEGIN CMNT;  
<CMNT>. |  
<CMNT>\n ;  
<CMNT>{commentMultEnd} BEGIN INITIAL; {printf("Multi line comment\n");}  
%%  
  
int main(int argc, char *argv)  
{  
    yylex();  
    return 0;  
}
```

Input file

```
//testing  
  
int main()  
{  
  
    /* multiline comment  
    continue...  
*/  
  
    /* Mult Com */  
  
    return 0;  
}
```

```
jara@jara-System-Product-Name:~/ETC/CompilerLab/Lab01/LexicalAnalysis/EX$ ./a.out < test.c  
Single line comment  
  
int main()  
{  
Multi line comment  
  
Multi line comment  
    return 0;  
}
```

소스코드 분석하기

- 해당 실습 목표
 - 소스코드를 읽어와 각 토큰을 타입과 함께 출력
- 언어의 명세
 - 다음과 같은 symbol을 가짐
 - + - * / () ; , .
 - 다음과 같은 연산자를 지원
 - =(EQ) <>(NOTEQ) <(LT) >(GT) <=(LE) >=(GE) :=(COLOEQ)
 - 다음과 같은 예약어 가짐
 - const, var, function, begin, end
 - if, then, while, do, return, write, writeln, odd
 - 식별자와 상수를 가질 수 있음
 - 식별자의 경우 영문자로 시작하며 숫자를 포함할 수 있음

결과 예시

Input

```
const n=5;  
var x;  
begin  
    x:=n;  
    write x;  
    writeln  
end.
```

Result


```
jara@jara-System-Product-Name:~/ETC/CompilerLab/Lab01.LexicalAnalysis/HW$ ./a.out < ex.pl0  
CONST : const  
Identifier : n  
EQ : =  
Number : 5  
Symbol : ;  
VAR : var  
Identifier : x  
Symbol : ;  
BEGIN : begin  
Identifier : x  
COLSEQ : :=  
Identifier : n  
Symbol : ;  
WRITE : write  
Identifier : x  
Symbol : ;  
WRITELN : writeln  
END : end  
Symbol : .  
symbolCount : 5  
digitCount : 1  
identCount : 5  
lineCount : 7
```

유의사항

- Symbol의 경우
 - Symbol : 해당 심볼
- 연산자의 경우
 - 연산자명 : 해당 연산자
- 예약어의 경우
 - 예약어명 : 해당 예약어
- 식별자의 경우
 - Identifier : 해당 식별자
- 상수의 경우
 - Number : 해당 숫자
- 출력 마지막에는 입력받은 Symbol, 상수, 식별자의 수와 소스코드의 라인 수를 출력하고 종료

유의사항

- 10월 5일 23시 59분 까지 제출
- Lex source code(.l)을 압축하여 다음과 같은 형식으로 제출
 - <학번>_Lab01.tar.gz
- 제출 기한 초과시 1일당 20% 감점
- tar 사용법
 - 압축
 - \$ tar -zcvf <output_file><input_file>
 - Ex) tar -zcvf 20202222_Lab01.tar.gz hw.l
 - 압축 해제
 - \$ tar -zxvf <input_file>
 - Ex) tar -zxvf ex.tar.gz



Q&A