
Using Flex

Introduction

Flex is a tool for generating programs that perform pattern-matching on text. This report will be a step-by-step tutorial for using Flex on Ubuntu to generate a simple lexical analyzer according to a set of given rules.

Installing Flex

```
apt-get update
apt-get install flex
```

Rules File

Flex takes as input a rules file that contains a set of regular expressions with associated blocks of code which will be executed when a match is found for the relevant regular expression.

Following is the rules file used to generate the required lexical analyzer with commented descriptions for each portion of the file:

```
// lex_rules.l

// Definitions section: This part of the file contains some definitions which can
// be used later in the rules section

// A definition consists of a name and a regular expression
DIGIT      [0-9]

// A definition can be used by enclosing it between {}
// The following line is translated to:
// DIGITS  ([0-9])+
DIGITS     {DIGIT}+

LETTER     [a-zA-Z]

ID         {LETTER} ({LETTER} | {DIGIT}) *

NUM        {DIGITS} | {DIGITS} \. {DIGITS} (E {DIGITS}) {0,1}

RELOP      \=|\=|!\=|>|>|\=|<|<|\=

ASSIGN     \=
```

```

PRIMITIVE  boolean|int|float

FLOW      if|else|while

PUNCT     \;|\,|\(|\)|\[[|\]|\\{|\\}

ADDOP     \+|\-

MULOP     \*|\\

%%  // Here starts the Rules Section

// The rules section contains a set of rules with their associated code blocks.
// A rule consists of a regular expression followed by a line of code or code
block.

// The regular expression here is using a definition PRIMITIVE
// When that regular expression is matched in the input the followin line of code
is executed where yytext is the matched input string.
{PRIMITIVE}    printf( "%s\n", yytext );

{FLOW}         printf( "%s\n", yytext );

{ID}           printf( "id\n");
{NUM}          printf( "num\n" );
{RELOP}        printf( "relop\n");

{ASSIGN}       printf( "%s\n", yytext );
{PUNCT}        printf( "%s\n", yytext );
{ADDOP}        printf( "addop\n" );
{MULOP}        printf( "mulop\n");

.              /*do nothing*/

%% // Here starts the Code Section

// This section contains the rest of the code of the lexical analyzer
// As we see here the program takes a file as its argument and opens the file for
reading.
// yylex() is the actual lexical analyzer created by flex which will scan the input
trying to match any of the specified rules

int main( int argc, char **argv )
{
    ++argv, --argc;      /* skip over program name */
    if ( argc > 0 )
        yyin = fopen( argv[0], "r" );
    else
        yyin = stdin;

    yylex();
}

```

Run Flex

Now you can run flex passing the rules file as an input and the name of the output file. See the following command:

```
flex lex_rules.l lexer.cpp
```

The output file is the required lexical analyzer according to the rules file. It can now be compiled to get the lexical analyzer executable.

We are using CMake for building the project. Running flex is included as a step of the building process. Run the following command to prepare the project:

```
cmake ./
```

Then run the following command to run flex and build the lexical analyzer:

```
cmake --build ./
```

Running Lexical Analyzer

The generated lexical analyzer is found in ./build/bin/LexicalAnalyzerFlex which can be used as follows:

```
./build/bin/LexicalAnalyzerFlex prog
```

Where *prog* is a program passed to the lexical analyzer in order to be scanned.

Here is the contents of *prog* file:

```
int sum , count , pass , mnt; while (pass != 10)
{
pass = pass + 1 ;
}
```

And this is the output of the generated lexical analyzer:

```
int
id
,
id
,
id
,
id
;
while
```

```
(  
id  
relop  
num  
)
```

```
{  
  
id  
=  
id  
addop  
num  
;  
  
}
```

Screenshots

- Regular Expressions

```
lex_rules.l (~/.javacompiler/flex) - gedit
Open Save Print Undo Redo
CMakeLists.txt x lex_rules.l x README.md x
DIGIT      [0-9]
DIGITS     {DIGIT}+
LETTER     [a-zA-Z]

ID         {LETTER}({LETTER}|{DIGIT})*

NUM        {DIGITS}|{DIGITS}\.{DIGITS}(E{DIGITS}){0,1}
|
RELOP      \=\=|!\=|>|>\=|<|<\=

ASSIGN     \=

PRIMITIVE  boolean|int|float

FLOW       if|else|while

PUNCT      \;|\,|\(|\)|\[\|\]\|\{\|\}

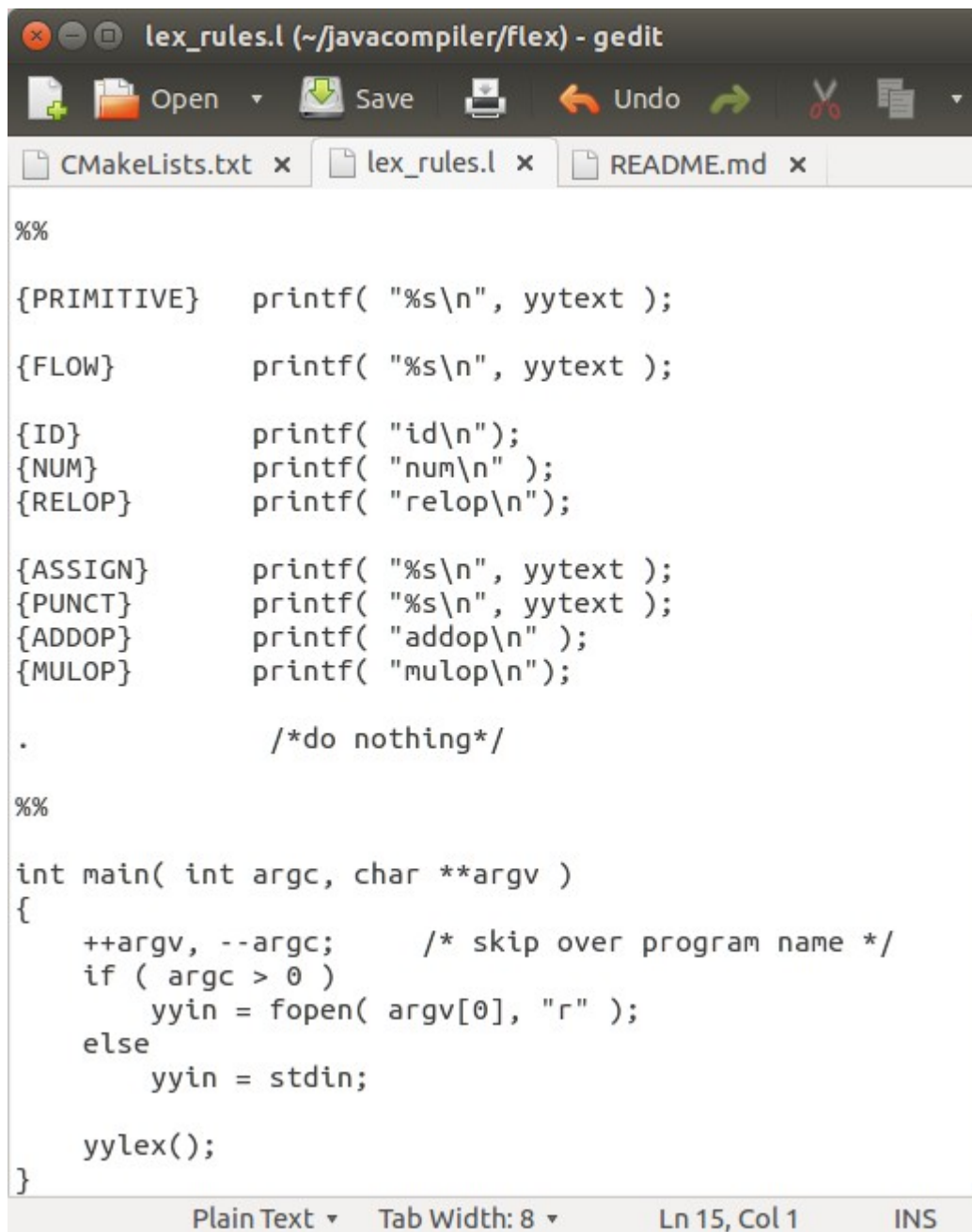
ADDOP      \+|\-

MULOP      \*|\\

%%

{PRIMITIVE} printf( \"%s\\n\", yytext );
Plain Text Tab Width: 8 Ln 9, Col 1 INS
```

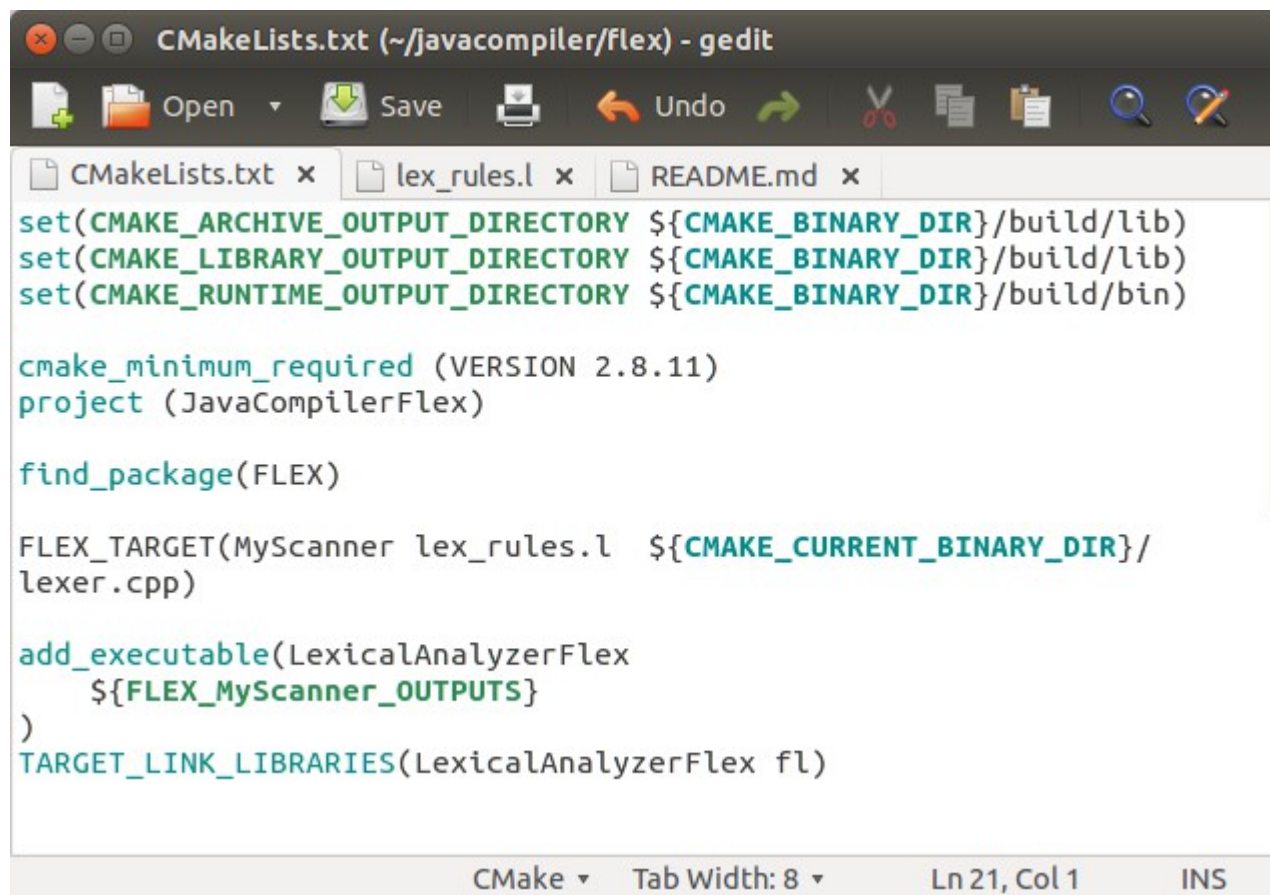
– Rules & code



```
%%  
  
{PRIMITIVE}    printf( "%s\n", yytext );  
  
{FLOW}         printf( "%s\n", yytext );  
  
{ID}           printf( "id\n");  
{NUM}          printf( "num\n" );  
{RELOP}        printf( "relop\n");  
  
{ASSIGN}       printf( "%s\n", yytext );  
{PUNCT}        printf( "%s\n", yytext );  
{ADDOP}        printf( "addop\n" );  
{MULOP}        printf( "mulop\n");  
  
.  
                /*do nothing*/  
  
%%  
  
int main( int argc, char **argv )  
{  
    ++argv, --argc;    /* skip over program name */  
    if ( argc > 0 )  
        yyin = fopen( argv[0], "r" );  
    else  
        yyin = stdin;  
  
    yylex();  
}
```

Plain Text ▾ Tab Width: 8 ▾ Ln 15, Col 1 INS

-
- CMake file to run Flex and build the generated Lexical Analyzer



```
set(CMAKE_ARCHIVE_OUTPUT_DIRECTORY ${CMAKE_BINARY_DIR}/build/lib)
set(CMAKE_LIBRARY_OUTPUT_DIRECTORY ${CMAKE_BINARY_DIR}/build/lib)
set(CMAKE_RUNTIME_OUTPUT_DIRECTORY ${CMAKE_BINARY_DIR}/build/bin)

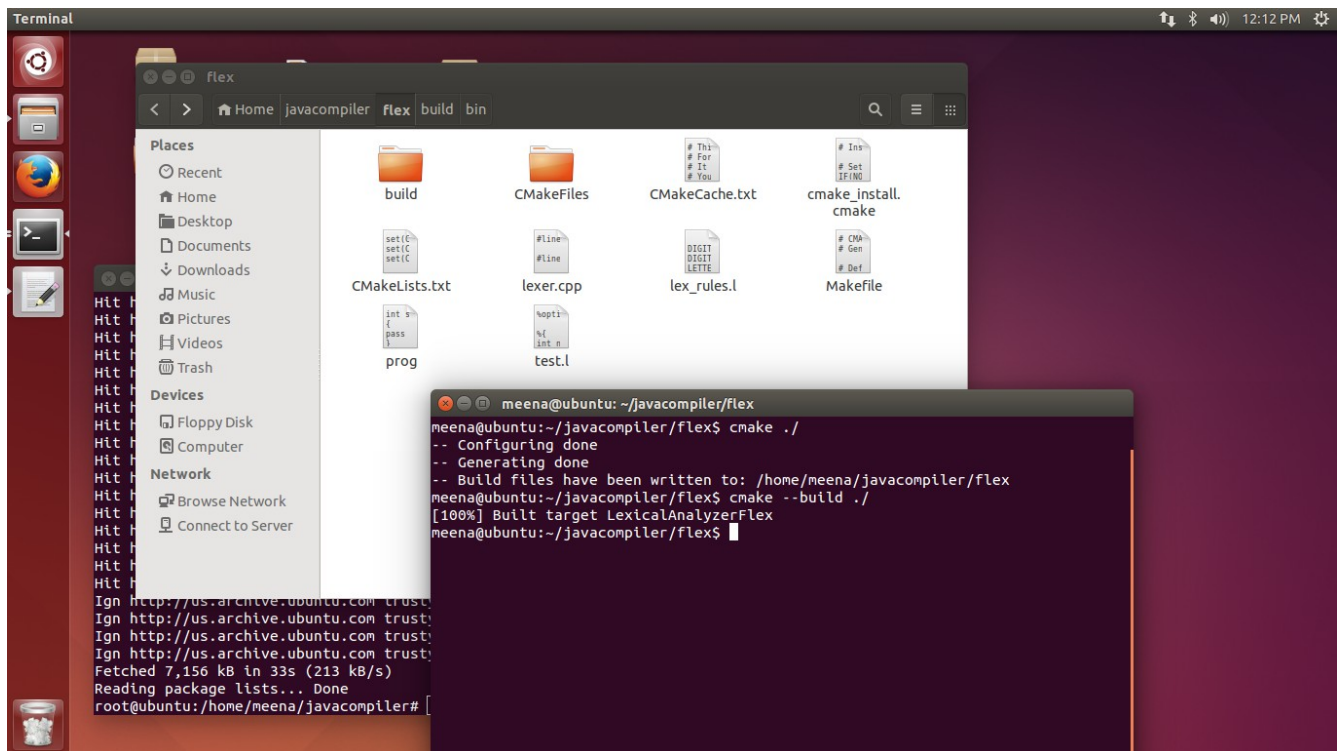
cmake_minimum_required (VERSION 2.8.11)
project (JavaCompilerFlex)

find_package(FLEX)

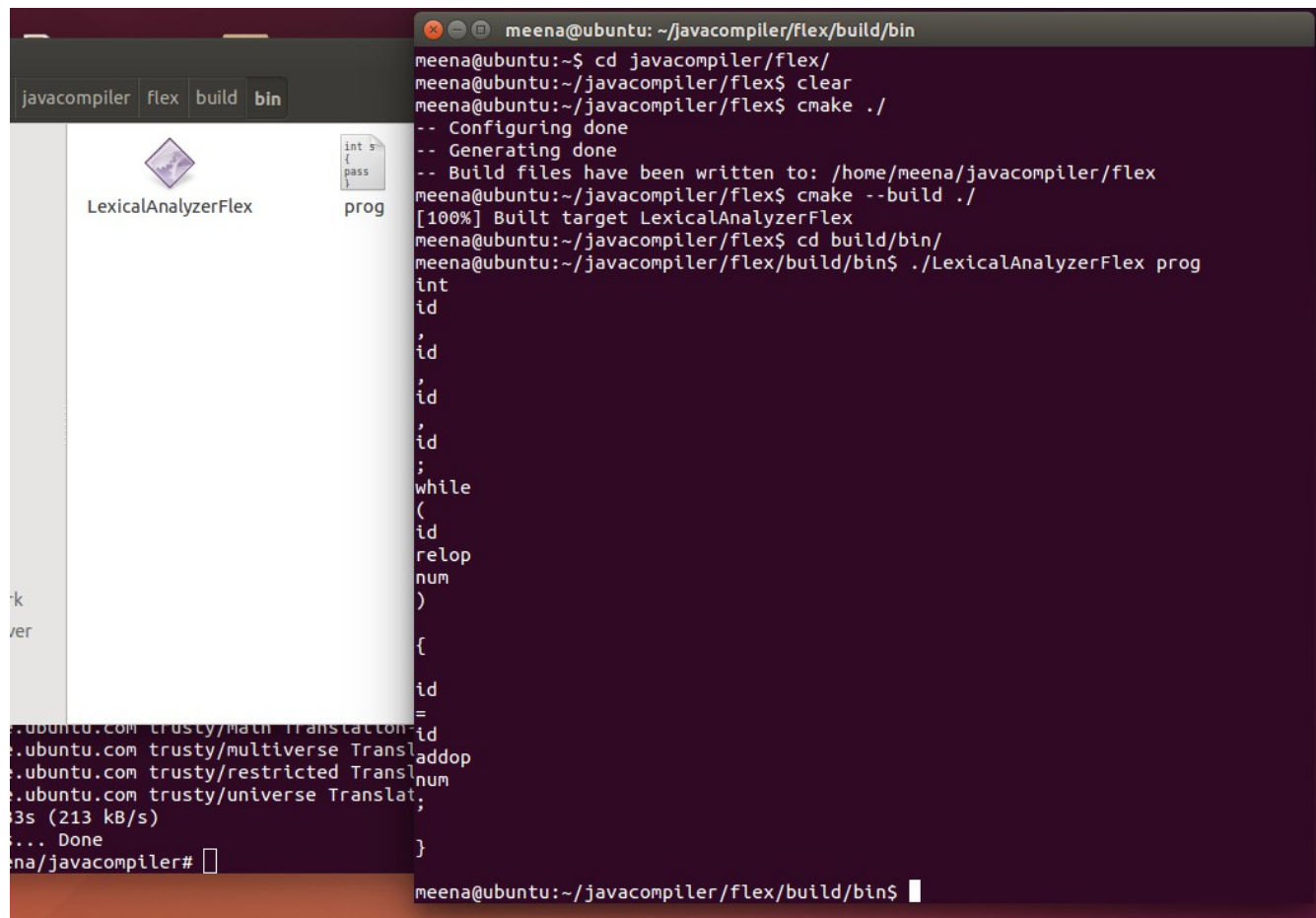
FLEX_TARGET(MyScanner lex_rules.l ${CMAKE_CURRENT_BINARY_DIR}/
lexer.cpp)

add_executable(LexicalAnalyzerFlex
    ${FLEX_MyScanner_OUTPUTS}
)
TARGET_LINK_LIBRARIES(LexicalAnalyzerFlex fl)
```

- Building project with cmake.
 - This will run Flex creating lexer.cpp from lex_rules.l
 - Then the lexical analyzer code will be built into LexicalAnalyzerFlex executable



- Here LexicalAnalyzerFlex is run on the prog file containing the example c program.



The screenshot shows a terminal window with the title bar "meena@ubuntu: ~/javacompiler/flex/build/bin". The terminal output is as follows:

```
meena@ubuntu:~$ cd javacompiler/flex/
meena@ubuntu:~/javacompiler/flex$ clear
meena@ubuntu:~/javacompiler/flex$ cmake .
-- Configuring done
-- Generating done
-- Build files have been written to: /home/meena/javacompiler/flex
meena@ubuntu:~/javacompiler/flex$ cmake --build ./
[100%] Built target LexicalAnalyzerFlex
meena@ubuntu:~/javacompiler/flex$ cd build/bin/
meena@ubuntu:~/javacompiler/flex/build/bin$ ./LexicalAnalyzerFlex prog
int
id
,
id
,
id
,
id
;
while
(
id
relop
num
)
{
id
=
id
addop
num
;
}
meena@ubuntu:~/javacompiler/flex/build/bin$
```

On the left side of the terminal, there is a file manager window showing the contents of the "javacompiler" directory. It contains subdirectories "flex", "build", and "bin". The "bin" directory is selected, showing a file named "LexicalAnalyzerFlex" and a file named "prog". The "prog" file is highlighted, showing its contents:

```
int
{
pass
}
```