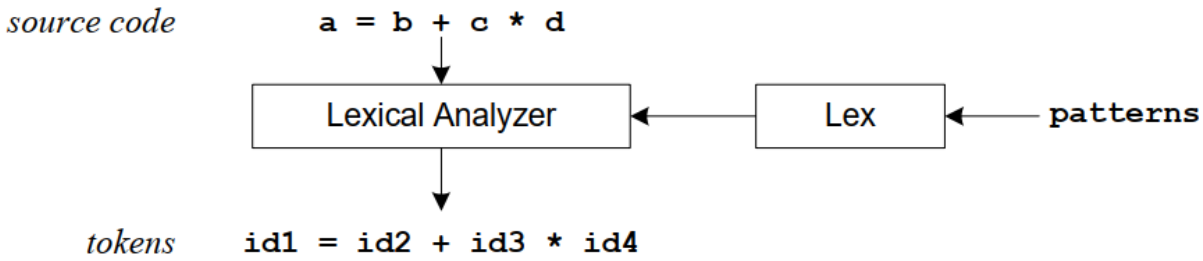
University of Asia Pacific
Department of Computer Science & Engineering
CSE 430: Compiler Design Lab

Task: Create a lexical analyzer using lex on Linux environment which will be able to recognize keywords, digits, identifiers and comments

Lex is a tool or software which automatically generates a lexical analyzer (finite Automata). It takes as its input a LEX source program and produces lexical Analyzer as its output. Lexical Analyzer will convert the input string entered by the user into tokens as its output. We define regular expressions by using lex. The following is the process where the source code is converted into tokens by using lex tool:



File format of Lex

```
{ definitions }
%%
 { rules }
%%
{ user subroutines }
```

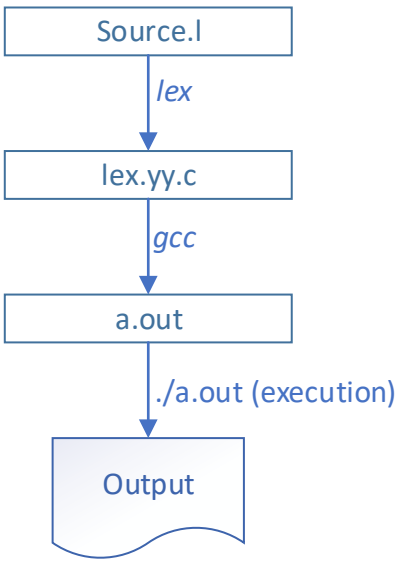
Definitions include declarations of constant, variable and regular definitions.

Rules define the statement of form p1 {action1} p2 {action2}....pn {action}.

Where **p** describes the regular expression and **action** describes the actions what action the lexical analyzer should take when pattern pi matches a lexeme.

Function of Lex

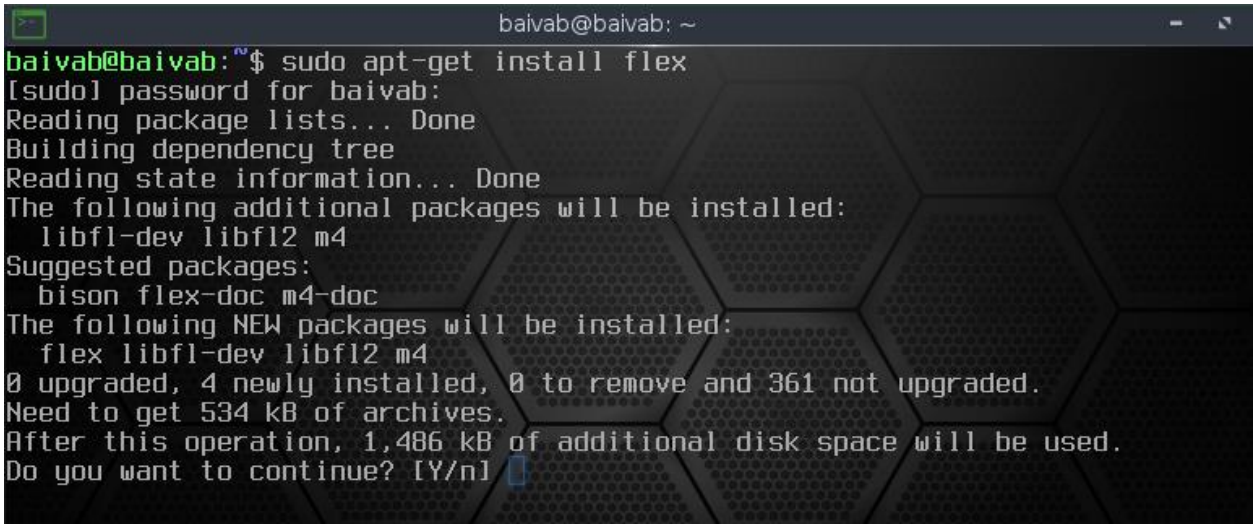
The process of compiling a lex program is show below:



Installation of Lex

Assuming the student has access to an updated Linux environment (preferably Debian based systems), follow the process shown below to install flex:

Open terminal and type "sudo apt-get install flex". A prompt will occur asking permission to download the flex package. Press 'Y' and the installation will be completed automatically.



Compiling lex programs

Follow the process to compile a lex program to recognize patterns:

1. The regular expression of an identifier is:

   *letter(letter|digit)**

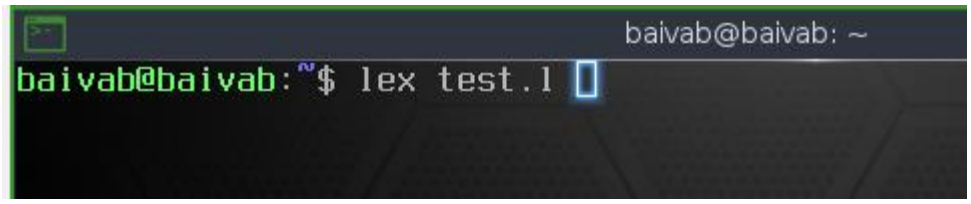
   We will use this regular expression in out lex program.

2. Create a file with '.l' extension such as: test.l and write the following code:

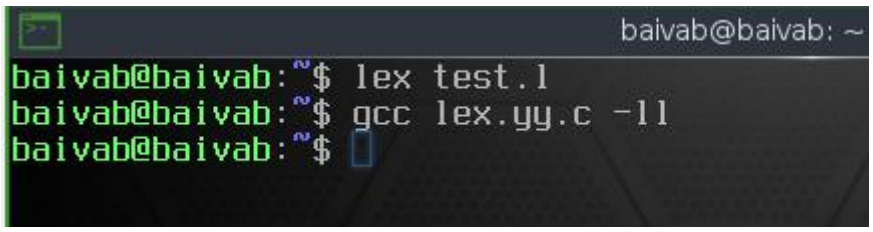
```
%{
#include<stdio.h>
%}

%%

 [a-zA-Z][a-zA-Z0-9]*  {printf ("Identifier \n");}

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

Here, the regular expression [a-zA-Z][a-zA-Z0-9]* is used for recognizing identifiers and the function yylex() returns a value indicating the type of token that has been obtained.

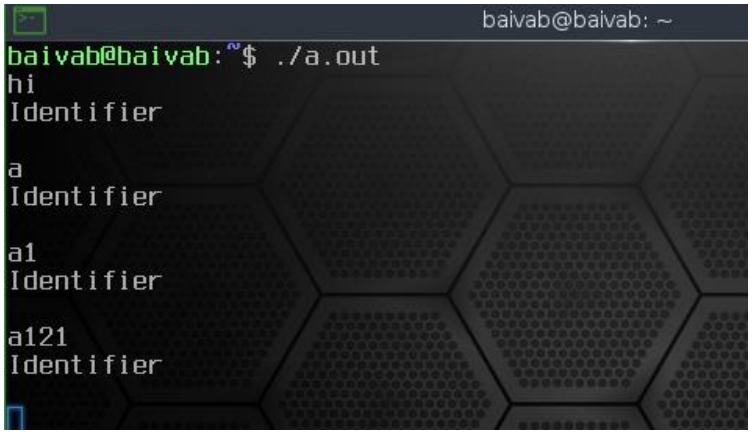
3. This test.l file will be now converted to a C program file by using the lex tool. Open terminal and change directory to the same directory where the test.l file is saved.
4. Type lex test.l and a new file name lex.yy.c will be created.



5. We can observe that this is a C file. So, we will use the gcc compiler to compile this c program. Type the command "gcc lex.yy.c -ll" to compile. A new file 'a.out' will be created which is the executable machine code file.



6. Now we execute the 'a.out' executable file to run the program by typing "./a.out"



7. We can see by giving input which matches with the regular expression of identifier it is recognizing the identifiers.

Pattern Matching Primitives

Metacharacter	Matches
.	any character except newline
\n	newline
*	zero or more copies of the preceding expression
+	one or more copies of the preceding expression
?	zero or one copy of the preceding expression
^	beginning of line
$	end of line
a|b	a or b
(ab)+	one or more copies of ab (grouping)
"a+b"	literal "a+b" (C escapes still work)
[]	character class

Pattern Matching Examples

Expression	Matches
abc	abc
abc*	ab abc abcc abccc ...
abc+	abc abcc abccc ...
a(bc)+	abc abcbc abcbcbc ...
a(bc)?	a abc
[abc]	one of: a, b, c
[a-z]	any letter, a-z
[a\-z]	one of: a, -, z
[-az]	one of: -, a, z
[A-Za-z0-9]+	one or more alphanumeric characters
[\t\n]+	whitespace
[^ab]	anything except: a, b
[a^b]	one of: a, ^, b
[a|b]	one of: a, |, b

a|b	one of: a, b

Sample Input

```
void main(){ int a, b2, c;   //hello    a = b2 * c + 10; }
```

Sample Output

```
void        keyword
main        identifier
int         keyword
a           identifier
b2          identifier
c           identifier
//hello     comment
10          digit
```