FLEX/BISON

LEXICAL ANALYZERS

BY

Mayar El Mahdy – 4639

EL-Zahraa Emara – 4558

Introduction:

Flex (fast lexical analyzer generator) , is a tool/computer program for generating lexical analyzers (scanners or lexers) , It is a free implementation of lex UNIX, Instead of writing a scanner from scratch, you only need to identify the *vocabulary* of a certain language (e.g. Simple), write a specification of patterns using regular expressions (e.g. DIGIT [0-9]), and FLEX will construct a scanner for you.

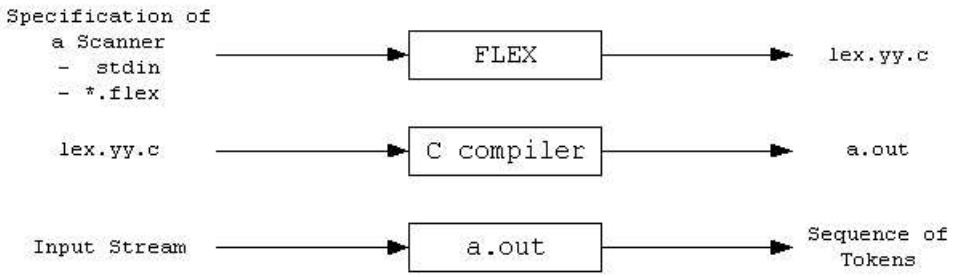

Figure-Steps to use flex on linux

Files Structure in flex:

1- Input file (file.l)

Your input file is consisting of three parts

Definitions

%%%

Rules

%%%

User code

Each part is separated by (%%%) , the first and last part can be left empty.

The middle part has the regular expression rules that your scanner – Flex- would use on your code that is provided.

1. Definition Section: The definition section contains the declaration of variables, regular definitions, manifest constants. In the definition section, text is enclosed in **"%{ %}"** brackets. Anything written in this brackets is copied directly to the file lex.yy.c

2. Rules Section: The rules section contains a series of rules in the form: *pattern action* and pattern must be unintended and action begin on the same line in {} brackets. The rule section is enclosed in **"%%** %%".

3. User Code Section: This section contain C statements and additional functions. We can also compile these functions separately and load with the lexical analyzer.

2- lex.yy.c

This is the file that defines a routine *yylex()* that uses the specification to recognize tokens , this is generated by the flex.

3-a.out

This is the actual scanner.

Steps to use Flex:

1-You need to make a new txt file and save it as filename.l , in this file you should follow the syntax I mentioned above on how to write in this file.

2- To compile it open the command window and type

Flex filename.l

3- A new file named lex.yy.c will be created , you should compile it by to generate the a.out file

gcc lex.yy.c

3- Finally you should compile the a.out file by

./a.out

Sample runs :

```
/*** Definition Section has one variable
which can be accessed inside yylex()
and main() ***/
%{
int count = 0;
%}

/*** Rule Section has three rules, first rule
matches with capital letters, second rule
matches with any character except newline and
third rule does not take input after the enter***/
%%
[A-Z] {printf("%s capital letter\n", yytext);
        count++;}
.        {printf("%s not a capital letter\n", yytext);}
\n {return 0;}
%%

/*** Code Section prints the number of
capital letter present in the given input***/
int yywrap(){}
int main(){
yylex();
printf("\nNumber of Captial letters "
        "in the given input - %d\n", count);

return 0;
}
```

```
mayar@mayar-VirtualBox:~$ lex file.l
mayar@mayar-VirtualBox:~$ gcc lex.yy.c
mayar@mayar-VirtualBox:~$ ./a.out
mayar
m not a capital letter
a not a capital letter
y not a capital letter
a not a capital letter
r not a capital letter

Number of Captial letters in the given input - 0
mayar@mayar-VirtualBox:~$
```

```
%{
%}

%%
[0-9] {printf("\nA digit\n");}
int {printf("\nThis is integer");}
;  {printf(";");}
\n  {return 0;}
%%
int yywrap(){}
int main(){
yylex();

return 0;
}
```

```
mayar@mayar-VirtualBox:~$ lex file.l
mayar@mayar-VirtualBox:~$ gcc lex.yy.c
mayar@mayar-VirtualBox:~$ ./a.out
int 999;

This is integer
A digit

A digit

A digit
;mayar@mayar-VirtualBox:~$
```