WEEK-2: Implement the lexical analyzer using lex.

ALGORITHM:

Step1: Lex program contains three sections: definitions, rules, and user subroutines. Each section must be separated from the others by a line containing only the delimiter, %%. The format is as follows: definitions %% rules %% user_subroutines

Step2: In definition section, the variables make up the left column, and their definitions make up the right column. Any C statements should be enclosed in %{..}%. Identifier is defined such that the first letter of an identifier is alphabet and remaining letters are alphanumeric.

Step3: In rules section, the left column contains the pattern to be recognized in an input file to yylex(). The right column contains the C program fragment executed when that pattern is recognized. The various patterns are keywords, operators, new line character, number, string, identifier, beginning and end of block, comment statements, preprocessor directive statements etc.

Step4: Each pattern may have a corresponding action, that is, a fragment of C source code to execute when the pattern is matched.

Step5: When yylex() matches a string in the input stream, it copies the matched text to an external character array, yytext, before it executes any actions in the rules section.

Step6: In user subroutine section, main routine calls yylex(). yywrap() is used to get more input.

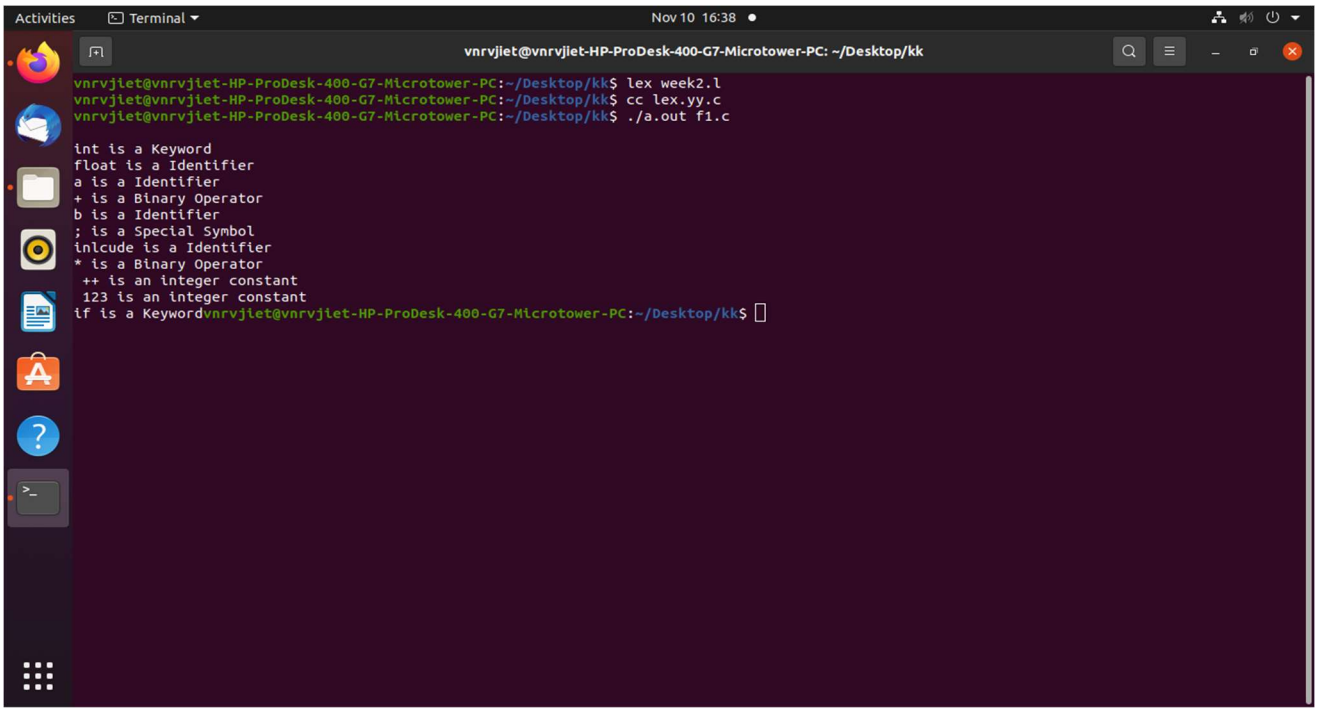
Step7: The lex command uses the rules and actions contained in file to generate a program, lex.yy.c, which can be compiled with the cc command. That program can then receive input, break the input into the logical pieces defined by the rules in file, and run program fragments contained in the actions in file.

```
%{
#include<stdio.h>
%}
digit [0-9]+
word [A-Za-z]+
spsym [(){};,%\{\]]
arith [+-/*]
whitspc[ \t\n]
underscr[_]
%%
{whitspc}+     ;

\"[^\n\"]*\"  {printf("\n %s is a literal",yytext);}
int |
include |
if |
else |
while |
do |
switch |
case |
default |
break |
continue |
scanf {printf("\n%s is a Keyword",yytext);}
{spsym} {printf("\n%s is a Special Symbol",yytext);}
{arith} {printf("\n%s is a Binary Operator",yytext);}
```

```
= {printf("\n%s is a Assignment operator",yytext);}
"++" |"--"   {printf("\n%s is an Unary Operator",yytext);}
"&" |"|" |"^"  {printf("\n %s is bitwise operator",yytext);}
"<" |">" |"<=" |">=" |"==" |"!="  {printf("\n %s is a relational
operator",yytext);}
{digit}+  {printf("\n %s is an integer constant",yytext);}
({digit}+)|({digit}*\.{digit}+)  {printf("\n %s is an float
constant",yytext);}
({word}({word}|{digit}|{underscr})*) {printf("\n%s is a
Identifier",yytext);}
%%
int main(int argc,char *argv[])
{
FILE *fp;
fp=fopen(argv[1],"r");
if(!fp)
{
printf("cnt open:%s",argv[1]);
exit(1);
}
yyin=fp;
yylex();

}
int yywrap()
{
return 1;
}
```

INPUT:

```
//var.c
#include<stdio.h>
#include<conio.h>
void main()
{
int a,b,c;
a=1;
b=2;
c=a+b;
printf("Sum:%d",c);
}
```


Activities
Terminal
Nov 10 16:38
vnrvjiet@vnrvjiet-HP-ProDesk-400-G7-Microtower-PC: ~/Desktop/kk
vnrvjiet@vnrvjiet-HP-ProDesk-400-G7-Microtower-PC:~/Desktop/kk$ lex week2.l
vnrvjiet@vnrvjiet-HP-ProDesk-400-G7-Microtower-PC:~/Desktop/kk$ cc lex.yy.c
vnrvjiet@vnrvjiet-HP-ProDesk-400-G7-Microtower-PC:~/Desktop/kk$ ./a.out f1.c
int is a Keyword
float is a Identifier
a is a Identifier
+ is a Binary Operator
b is a Identifier
; is a Special Symbol
inlcude is a Identifier
* is a Binary Operator
++ is an integer constant
123 is an integer constant
if is a Keywordvnrvjiet@vnrvjiet-HP-ProDesk-400-G7-Microtower-PC:~/Desktop/kk$