# LAB 2: 101485 David Ogaro

## Lab 2 Output Explanation.

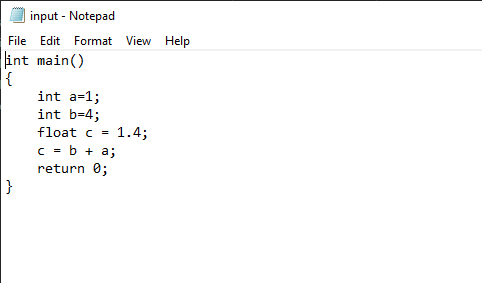
The scanner converts the high-level language input program into a sequence of tokens. The output is a sequence of tokens which is sent to the parser for syntax analysis. In Flex:

1. An input file describes the lexical analyzer to be generated named lex.l is written in lex language. The lex compiler transforms lex.l to C program, in a file that is always named lex.yy.c.
2. The C compiler compiles lex.yy.c file into an executable file called a.exe
3. The output file a.exe takes a stream of input characters and produces a stream of tokens.

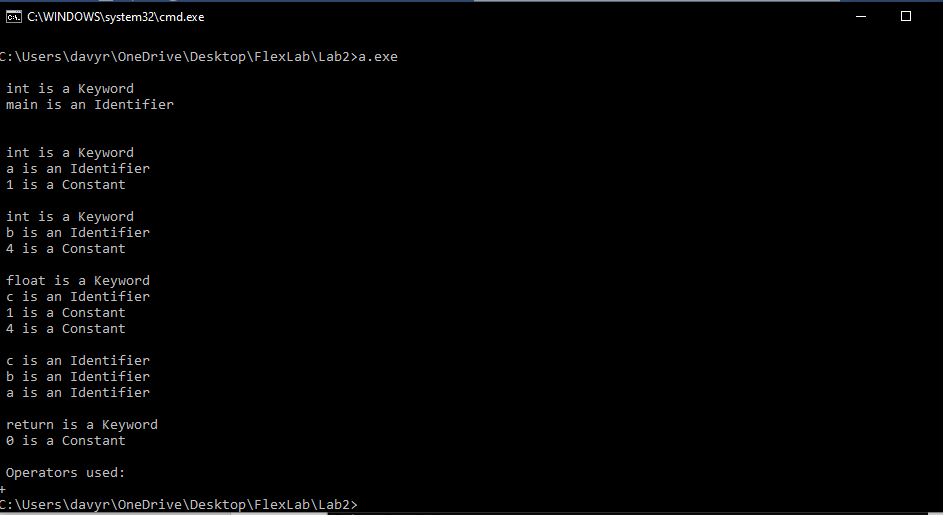
The Lexical analyzer:

1. Divides the program into valid tokens a process called Tokenization.
2. Removes white space characters.
3. Removes comments.
4. It also provides help in generating error messages by providing row numbers and column numbers.

## Input file



## Lab 2 Output



**From the text file, all the valid tokens are:**

‘int’ , ‘main’ , ‘(’ , ‘)’ , ‘{’ , ‘int’ , ‘a’ , ‘=’ , ‘1’ , ‘;’ , ‘int’ , ‘b’ , ‘=’ , ‘4’ , ‘;’ , ‘float’ , ‘c’ , ‘=’ , ‘1.4’ , ‘;’ , ‘c’ , ‘=’ , ‘b’ , ‘+’ , ‘a’ , ‘;’ , ‘return’ , ‘0’ , ‘;’ , ‘}’

Total tokens: 30

**The expression is then tokenized using the input file source program where:**

int is a keyword, main is an identifier

int is a keyword, a is an identifier, 1 is a constant

int is a keyword, b is an identifier, 4 is a constant

float is a keyword, c is an identifier, 1 is a constant, 4 is a constant

c is an identifier, b is an identifier, a is an identifier

return is a keyword, 0 is a constant

+ is an operator

The tokenized output is then displayed.