Flex (Fast Lexical Analyzer Generator)

Flex allows one to specify a lexical analyzer by specifying regular expressions to describe patterns for tokens. The input notation for the Lex tool is referred to as the Lex language and the tool itself is the Lex compiler. Behind the scenes, the Lex compiler transforms the input patterns into a transition diagram and generates code, in a file called **lex .yy. c**, that simulates this transition diagram.

The following figure illustrates the steps required to produce the Lexical Analyzer.

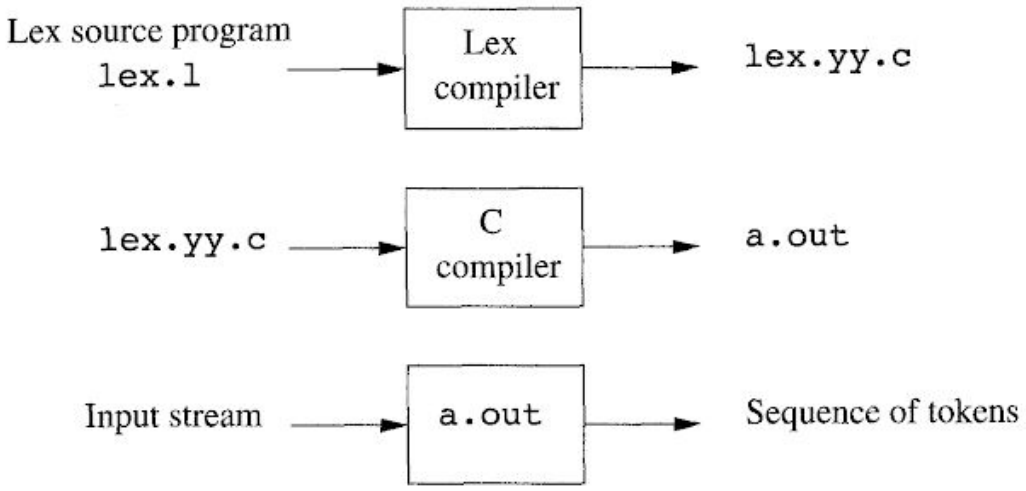


Figure 3.22: Creating a lexical analyzer with Lex from the reference.

Step 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.

Step 2:

Lex.yy.c is compiled by C compiler to produce an executable file a.out.

Step 3:

We can now use the executable file which takes a stream of input or reads an input file and outputs a stream of tokens.

Lex Program Structure:

The file is divided into 3 sections :

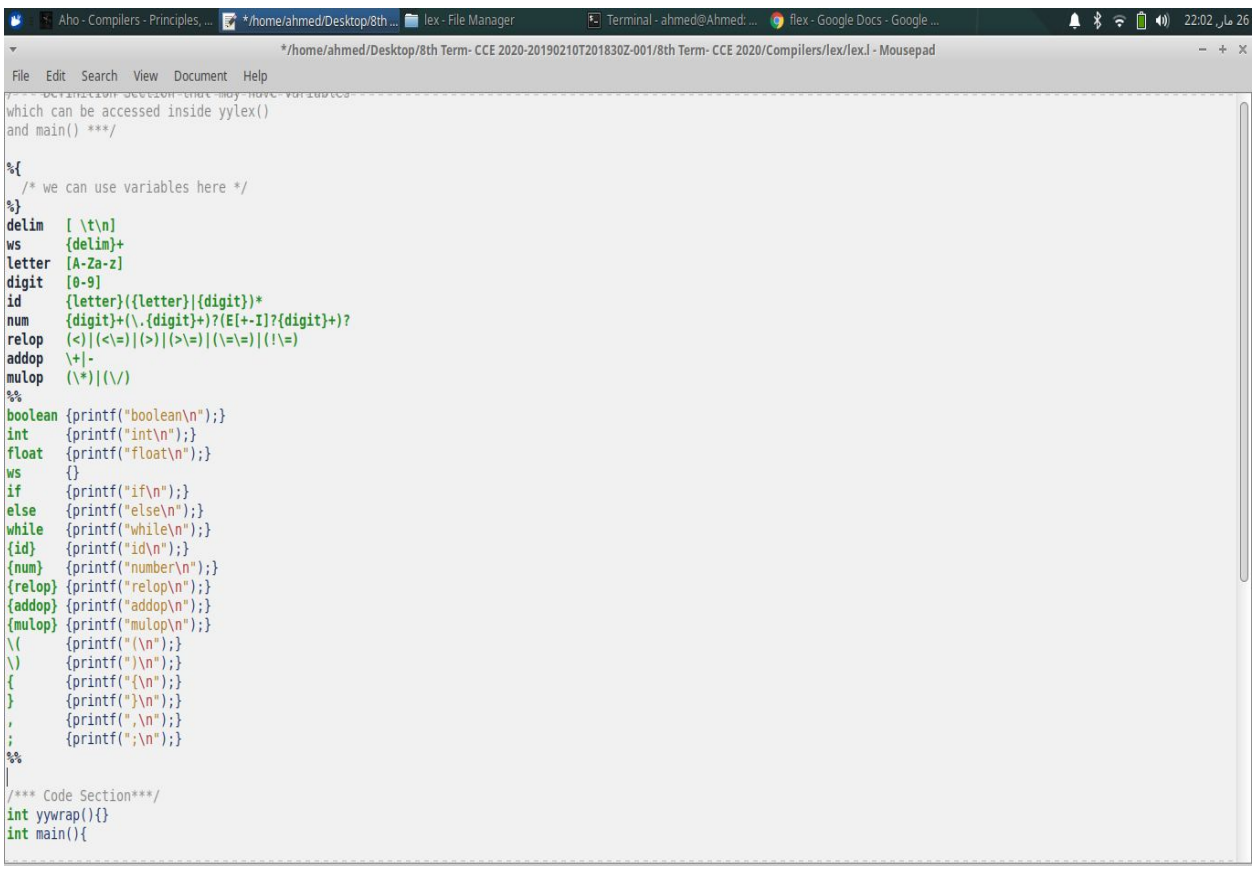
- **Declarations** : this section includes declarations of variables, constants and regular definitions.

- **Transition rules :** this section specifies the patterns, each pattern is a regex which may use the regular definitions in declarations section.

- **Auxiliary functions :** The third section holds whatever additional functions are used in the actions.

Screenshots:

lex program

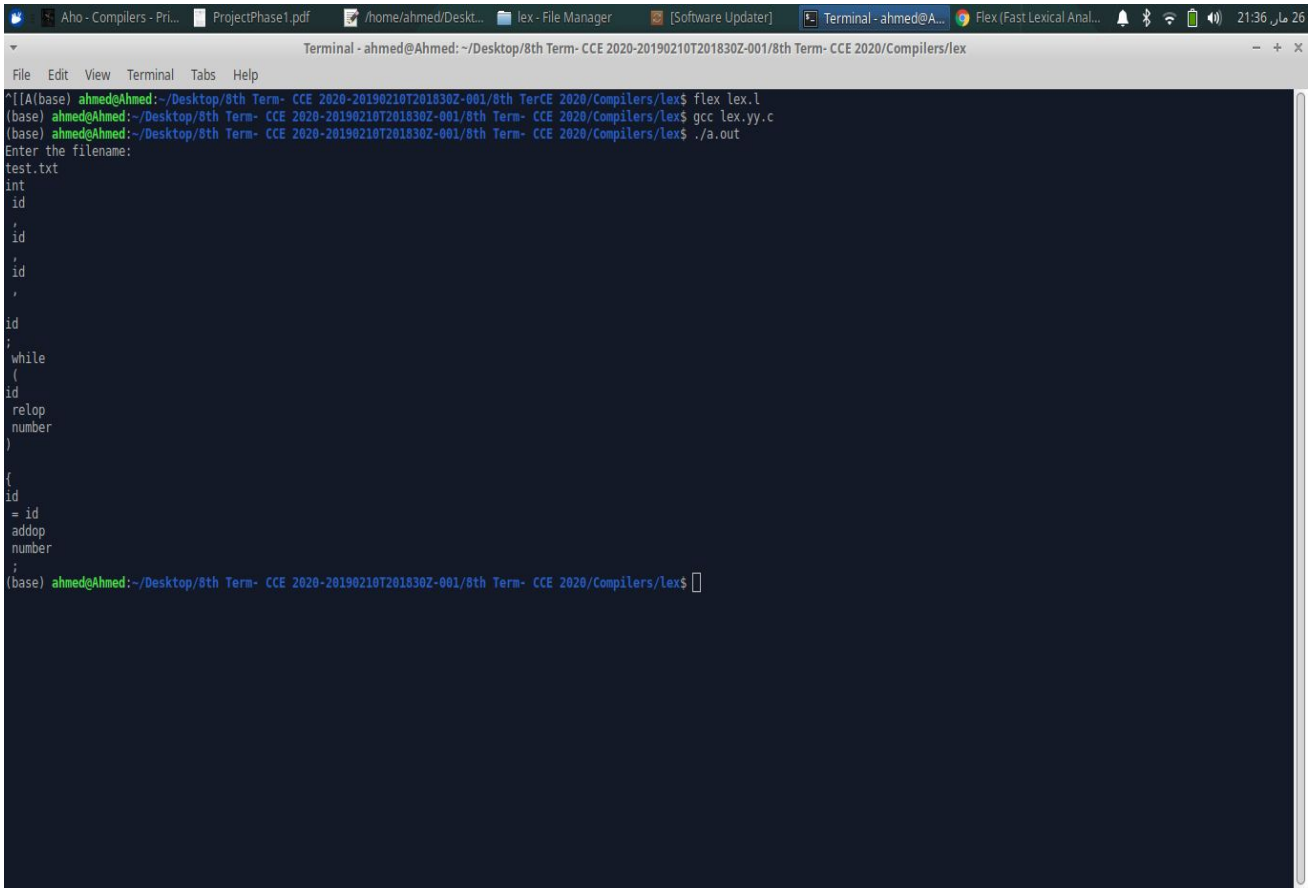



```
which can be accessed inside yylex()
and main() ***/

%{
  /* we can use variables here */
%}
delim   [ \t\n]
ws      {delim}+
letter  [A-Za-z]
digit   [0-9]
id      {letter}({letter}|{digit})*
num     {digit}+(\.{digit}+)?(E[+-I]?{digit}+)?
relop   (<)|(<\=)|(>)|(>\=)|(\=\=)|(!\=)
addop   \+|-
mulop   (\*)|(\/)
%%
boolean {printf("boolean\n");}
int     {printf("int\n");}
float   {printf("float\n");}
ws      {}
if      {printf("if\n");}
else    {printf("else\n");}
while   {printf("while\n");}
{id}    {printf("id\n");}
{num}   {printf("number\n");}
{relop} {printf("relop\n");}
{addop} {printf("addop\n");}
{mulop} {printf("mulop\n");}
\(      {printf("(\n");}
\)      {printf(")\n");}
{       {printf("{\n");}
}       {printf("}\n");}
,       {printf(",\n");}
;       {printf(";\n");}
%%

/*** Code Section***/
int yywrap(){}
int main(){
```

Sample run



The input file contains the same sample program as the on in the project PDF.