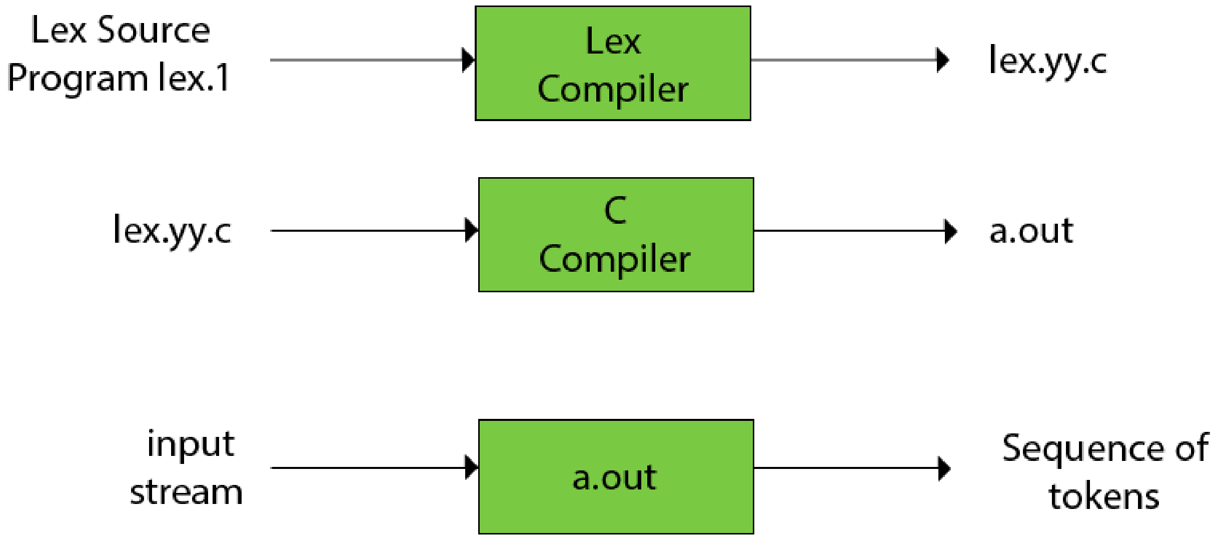
Bonus Bart

Lexical Analyzer Generator using Flex

LEX/FLEX:

Lexical analyzer generators, they are open-source tools, they are a computer programs that generates lexical analyzers.Flex is frequently used as the lex implementation but both of them have almost the same syntax(LEX is included in FLEX)

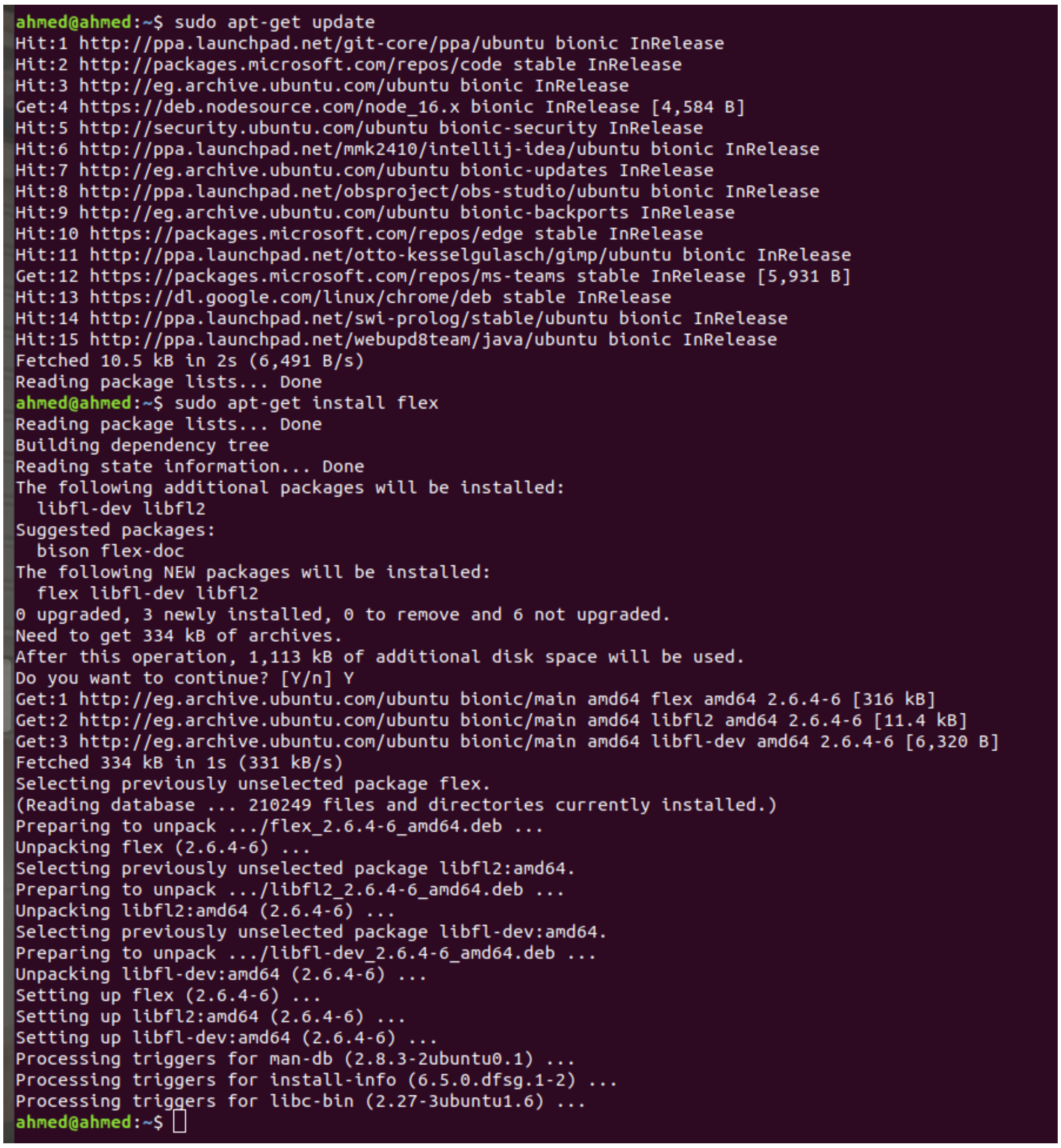
Functionality of LEX/FLEX:



We start by writing LEX/FLEX program, which is compiled by the LEX compiler to generate a .c file
The C file is compiled just like any normal code and get the executable .out file
Any input stream get into the executable file and generate sequence of tokens according to the rules that was defined in the source code

Steps to write and Run LEX/FLEX files:

1. **Install FLEX on the systems**
    a. For linux Ubuntu:
        Sudo apt-get update
        Sudo apt-get install flex



```
ahmed@ahmed:~$ sudo apt-get update
Hit:1 http://ppa.launchpad.net/git-core/ppa/ubuntu bionic InRelease
Hit:2 http://packages.microsoft.com/repos/code stable InRelease
Hit:3 http://eg.archive.ubuntu.com/ubuntu bionic InRelease
Get:4 https://deb.nodesource.com/node_16.x bionic InRelease [4,584 B]
Hit:5 http://security.ubuntu.com/ubuntu bionic-security InRelease
Hit:6 http://ppa.launchpad.net/mmk2410/intellij-idea/ubuntu bionic InRelease
Hit:7 http://eg.archive.ubuntu.com/ubuntu bionic-updates InRelease
Hit:8 http://ppa.launchpad.net/obsproject/obs-studio/ubuntu bionic InRelease
Hit:9 http://eg.archive.ubuntu.com/ubuntu bionic-backports InRelease
Hit:10 https://packages.microsoft.com/repos/edge stable InRelease
Hit:11 http://ppa.launchpad.net/otto-kesselgulasch/gimp/ubuntu bionic InRelease
Get:12 https://packages.microsoft.com/repos/ms-teams stable InRelease [5,931 B]
Hit:13 https://dl.google.com/linux/chrome/deb stable InRelease
Hit:14 http://ppa.launchpad.net/swi-prolog/stable/ubuntu bionic InRelease
Hit:15 http://ppa.launchpad.net/webupd8team/java/ubuntu bionic InRelease
Fetched 10.5 kB in 2s (6,491 B/s)
Reading package lists... Done
ahmed@ahmed:~$ sudo apt-get install flex
Reading package lists... Done
Building dependency tree
Reading state information... Done
The following additional packages will be installed:
  libfl-dev libfl2
Suggested packages:
  bison flex-doc
The following NEW packages will be installed:
  flex libfl-dev libfl2
0 upgraded, 3 newly installed, 0 to remove and 6 not upgraded.
Need to get 334 kB of archives.
After this operation, 1,113 kB of additional disk space will be used.
Do you want to continue? [Y/n] Y
Get:1 http://eg.archive.ubuntu.com/ubuntu bionic/main amd64 flex amd64 2.6.4-6 [316 kB]
Get:2 http://eg.archive.ubuntu.com/ubuntu bionic/main amd64 libfl2 amd64 2.6.4-6 [11.4 kB]
Get:3 http://eg.archive.ubuntu.com/ubuntu bionic/main amd64 libfl-dev amd64 2.6.4-6 [6,320 B]
Fetched 334 kB in 1s (331 kB/s)
Selecting previously unselected package flex.
(Reading database ... 210249 files and directories currently installed.)
Preparing to unpack .../flex_2.6.4-6_amd64.deb ...
Unpacking flex (2.6.4-6) ...
Selecting previously unselected package libfl2:amd64.
Preparing to unpack .../libfl2_2.6.4-6_amd64.deb ...
Unpacking libfl2:amd64 (2.6.4-6) ...
Selecting previously unselected package libfl-dev:amd64.
Preparing to unpack .../libfl-dev_2.6.4-6_amd64.deb ...
Unpacking libfl-dev:amd64 (2.6.4-6) ...
Setting up flex (2.6.4-6) ...
Setting up libfl2:amd64 (2.6.4-6) ...
Setting up libfl-dev:amd64 (2.6.4-6) ...
Processing triggers for man-db (2.8.3-2ubuntu0.1) ...
Processing triggers for install-info (6.5.0.dfsg.1-2) ...
Processing triggers for libc-bin (2.27-3ubuntu1.6) ...
ahmed@ahmed:~$
```

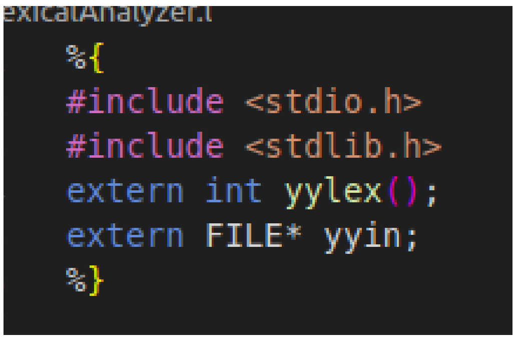
- Note: Installing flex including installing lex

2. **Write the program:**

   Lex code is divided into areas:

   a. **Header/import area:**

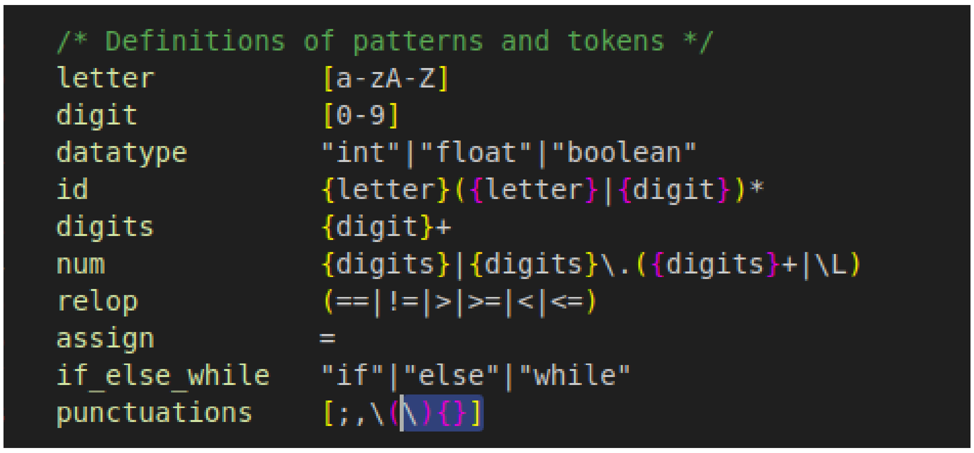
      Enclosed within%{.............%}, this area to include header files, declarations, and definitions that will be copied directly into the generated C code by Lex.



```
exicalAnalyzer.l
%{
#include <stdio.h>
#include <stdlib.h>
extern int yylex();
extern FILE* yyin;
%}
```

   b. **Definition/regular expression area:**

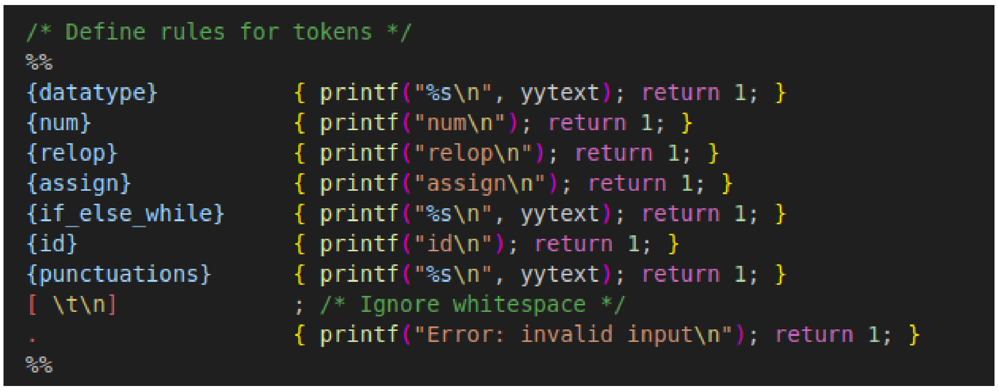
      define regular expressions that match various tokens from the input grammar



```
/* Definitions of patterns and tokens */
letter          [a-zA-Z]
digit           [0-9]
datatype        "int"|"float"|"boolean"
id              {letter}({letter}|{digit})*
digits          {digit}+
num             {digits}|{digits}\.({digits}+|\L)
relop           (==|!=|>|>=|<|<=)
assign          =
if_else_while   "if"|"else"|"while"
punctuations    [;,\(\){}]
```

   c. **Rules area:**

      Where we define the rules of each tokens and which action should be taken with each token



```
/* Define rules for tokens */
%%
{datatype}          { printf("%s\n", yytext); return 1; }
{num}               { printf("num\n"); return 1; }
{relop}             { printf("relop\n"); return 1; }
{assign}            { printf("assign\n"); return 1; }
{if_else_while}     { printf("%s\n", yytext); return 1; }
{id}                { printf("id\n"); return 1; }
{punctuations}      { printf("%s\n", yytext); return 1; }
[ \t\n]             ; /* Ignore whitespace */
.                   { printf("Error: invalid input\n"); return 1; }
%%
```

d. **Subroutine area:**
   Where we add auxiliary procedures needed by the actions. The subroutine can be loaded with the lexical analyzer and compiled separately. (we didn’t need any in our program).

How to run the code:

1. Open the terminal
2. cd “path to the file” → move to directory of the file
3. flex filename.l → compile lex to .c file
4. gcc -o outputfile_name file.c -ll → compile .c to get executable file
5. ./outputfile_name path to input file → run the code with input file

Test Case:

Using the following grammamr:

```
letter = a-z | A-Z
digit = 0 - 9
id: letter (letter|digit)*
digits = digit+
{boolean int float}
num: digit+ | digit+ . digits ( \L | E digits)
relop: \=\= | !\= | > | >\= | < | <\=
assign: =
{ if else while }
[; , \( \) { }]
addop: \+ | -
mulop: \* | /
```

With the following input:

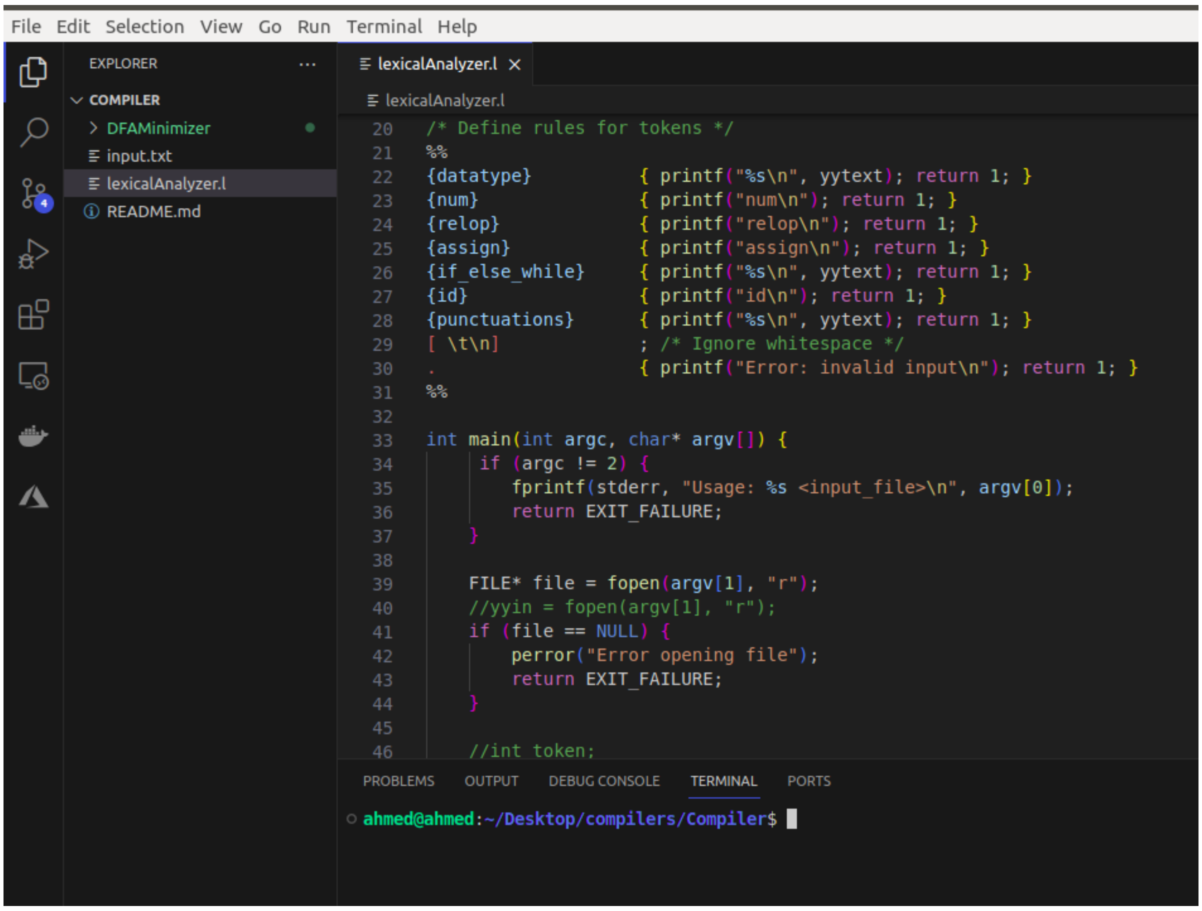
```
int sum , count , pass , mnt; while (pass !=10)
{
pass = pass + 1 ;
}
```

We get the following output:

```
int
id
,
id
,
id
,
id
;
while
(
id
relop
num
)
{
id
assign
id
Error: invalid input
num
;
}
```

Snipping:

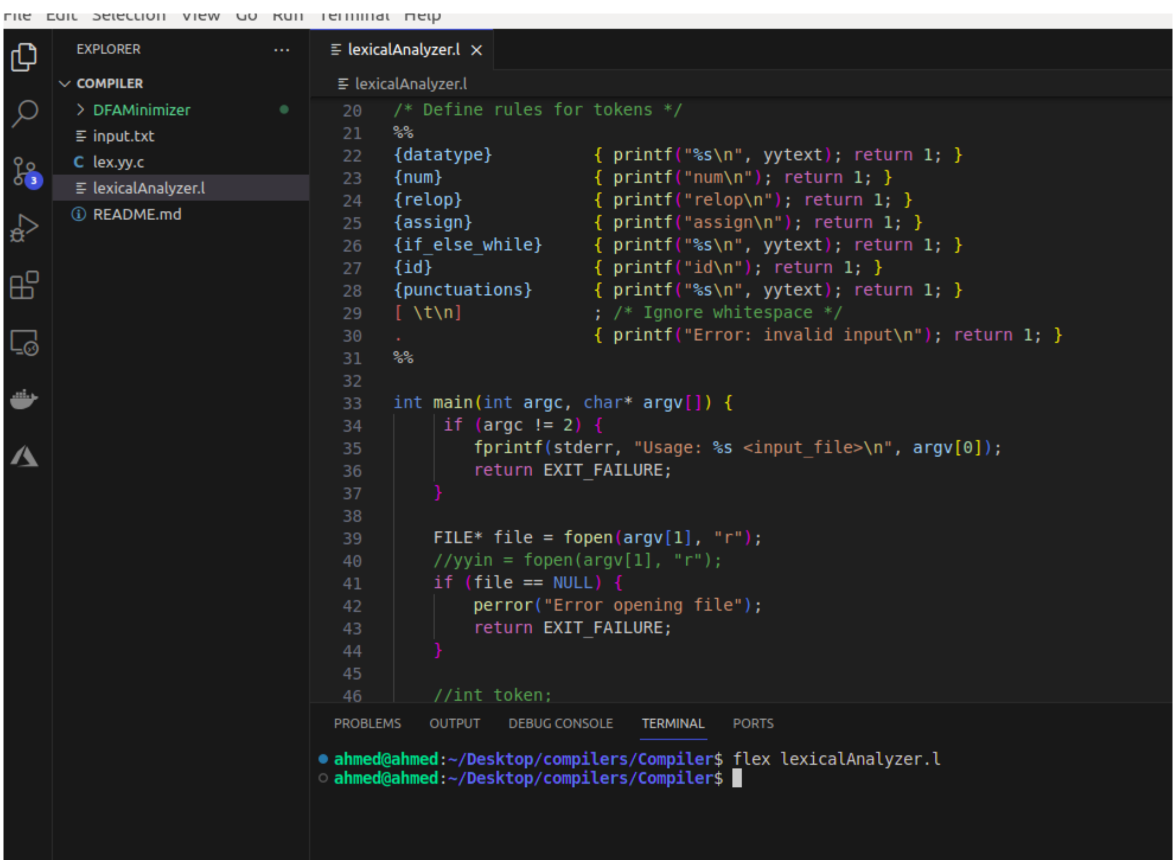
1. Before running any command:


File Edit Selection View Go Run Terminal Help
EXPLORER
COMPILER
DFAMinimizer
input.txt
lexicalAnalyzer.l
README.md
lexicalAnalyzer.l
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

```
/* Define rules for tokens */
%%
{datatype}          { printf("%s\n", yytext); return 1; }
{num}               { printf("num\n"); return 1; }
{relop}             { printf("relop\n"); return 1; }
{assign}            { printf("assign\n"); return 1; }
{if_else_while}     { printf("%s\n", yytext); return 1; }
{id}                { printf("id\n"); return 1; }
{punctuations}      { printf("%s\n", yytext); return 1; }
[ \t\n]             ; /* Ignore whitespace */
.                   { printf("Error: invalid input\n"); return 1; }
%%

int main(int argc, char* argv[]) {
    if (argc != 2) {
        fprintf(stderr, "Usage: %s <input_file>\n", argv[0]);
        return EXIT_FAILURE;
    }

    FILE* file = fopen(argv[1], "r");
    //yyin = fopen(argv[1], "r");
    if (file == NULL) {
        perror("Error opening file");
        return EXIT_FAILURE;
    }

    //int token;
```

```
ahmed@ahmed:~/Desktop/compilers/Compiler$
```

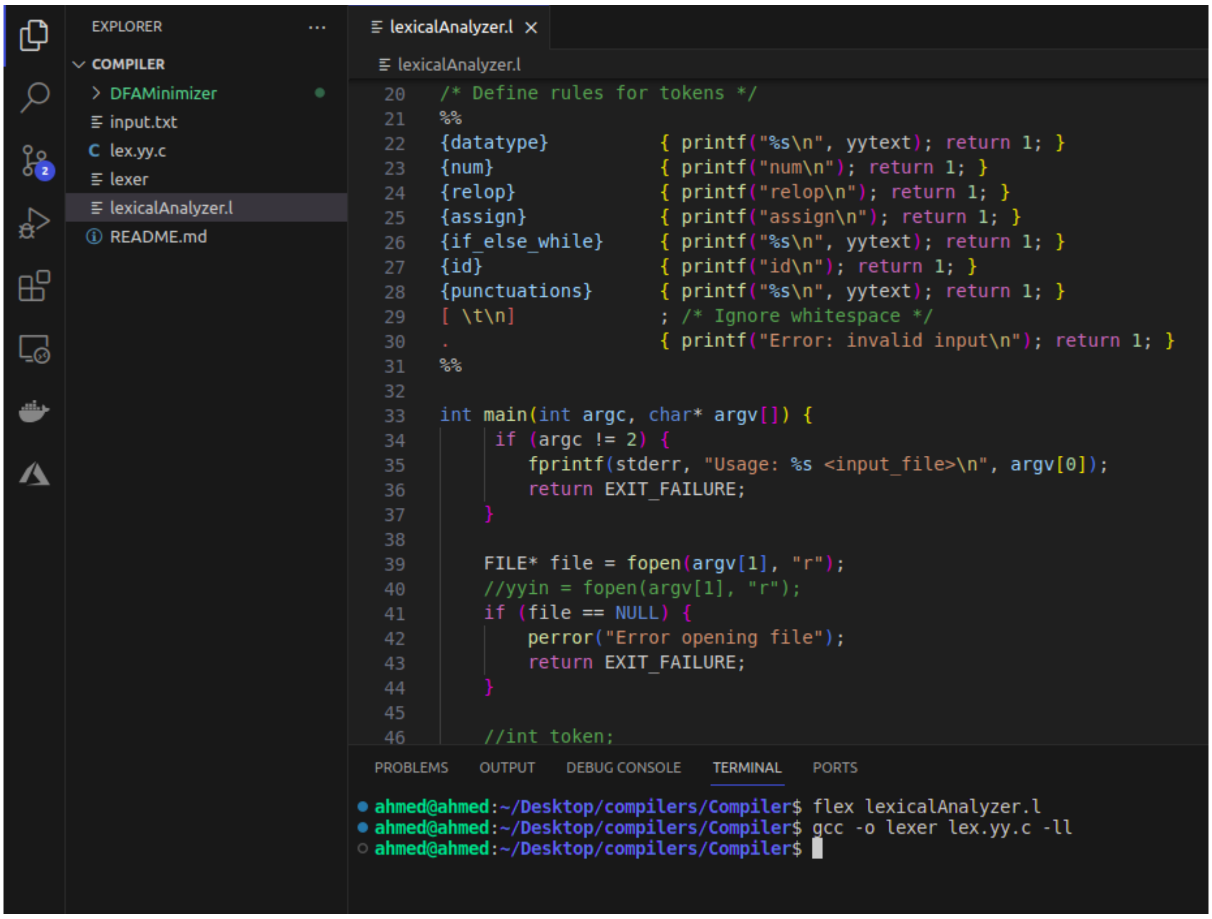
2. After flex command:


File Edit Selection View Go Run Terminal Help
EXPLORER
COMPILER
DFAMinimizer
input.txt
lex.yy.c
lexicalAnalyzer.l
README.md
lexicalAnalyzer.l
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

```
/* Define rules for tokens */
%%
{datatype}          { printf("%s\n", yytext); return 1; }
{num}               { printf("num\n"); return 1; }
{relop}             { printf("relop\n"); return 1; }
{assign}            { printf("assign\n"); return 1; }
{if_else_while}     { printf("%s\n", yytext); return 1; }
{id}                { printf("id\n"); return 1; }
{punctuations}      { printf("%s\n", yytext); return 1; }
[ \t\n]             ; /* Ignore whitespace */
.                   { printf("Error: invalid input\n"); return 1; }
%%

int main(int argc, char* argv[]) {
    if (argc != 2) {
        fprintf(stderr, "Usage: %s <input_file>\n", argv[0]);
        return EXIT_FAILURE;
    }

    FILE* file = fopen(argv[1], "r");
    //yyin = fopen(argv[1], "r");
    if (file == NULL) {
        perror("Error opening file");
        return EXIT_FAILURE;
    }

    //int token;
```

```
ahmed@ahmed:~/Desktop/compilers/Compiler$ flex lexicalAnalyzer.l
ahmed@ahmed:~/Desktop/compilers/Compiler$
```

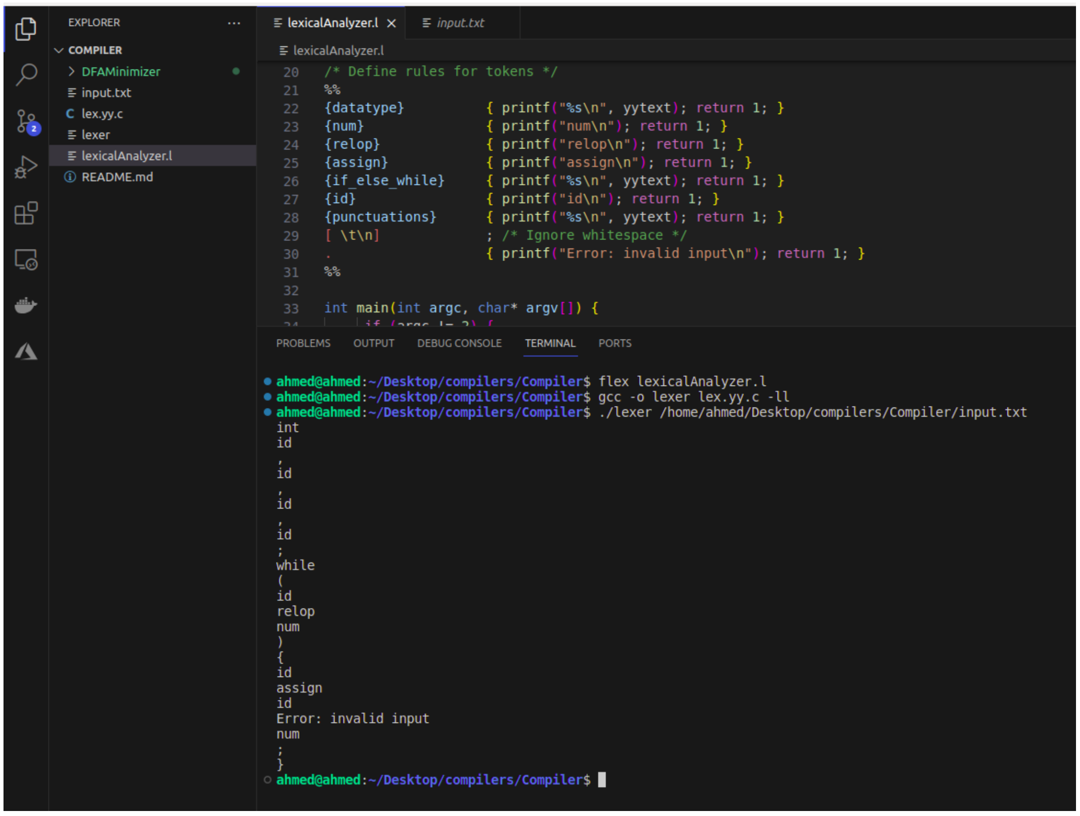
3. After compile .c file:



```
/* Define rules for tokens */
%%
{datatype}          { printf("%s\n", yytext); return 1; }
{num}               { printf("num\n"); return 1; }
{relop}             { printf("relop\n"); return 1; }
{assign}            { printf("assign\n"); return 1; }
{if_else_while}     { printf("%s\n", yytext); return 1; }
{id}                { printf("id\n"); return 1; }
{punctuations}      { printf("%s\n", yytext); return 1; }
[ \t\n]             ; /* Ignore whitespace */
.                   { printf("Error: invalid input\n"); return 1; }
%%

int main(int argc, char* argv[]) {
    if (argc != 2) {
        fprintf(stderr, "Usage: %s <input_file>\n", argv[0]);
        return EXIT_FAILURE;
    }

    FILE* file = fopen(argv[1], "r");
    //yyin = fopen(argv[1], "r");
    if (file == NULL) {
        perror("Error opening file");
        return EXIT_FAILURE;
    }

    //int token;
```

```
ahmed@ahmed:~/Desktop/compilers/Compiler$ flex lexicalAnalyzer.l
ahmed@ahmed:~/Desktop/compilers/Compiler$ gcc -o lexer lex.yy.c -ll
ahmed@ahmed:~/Desktop/compilers/Compiler$
```

4. The output:



```
/* Define rules for tokens */
%%
{datatype}          { printf("%s\n", yytext); return 1; }
{num}               { printf("num\n"); return 1; }
{relop}             { printf("relop\n"); return 1; }
{assign}            { printf("assign\n"); return 1; }
{if_else_while}     { printf("%s\n", yytext); return 1; }
{id}                { printf("id\n"); return 1; }
{punctuations}      { printf("%s\n", yytext); return 1; }
[ \t\n]             ; /* Ignore whitespace */
.                   { printf("Error: invalid input\n"); return 1; }
%%

int main(int argc, char* argv[]) {
```

```
ahmed@ahmed:~/Desktop/compilers/Compiler$ flex lexicalAnalyzer.l
ahmed@ahmed:~/Desktop/compilers/Compiler$ gcc -o lexer lex.yy.c -ll
ahmed@ahmed:~/Desktop/compilers/Compiler$ ./lexer /home/ahmed/Desktop/compilers/Compiler/input.txt
int
id
,
id
,
id
,
id
;
while
(
id
relop
num
)
{
id
assign
id
Error: invalid input
num
;
}
ahmed@ahmed:~/Desktop/compilers/Compiler$
```