Practical 1.

Aim:

Implement lexical analyzer to recognize all distinct token class (using Lex).

Code:

pract1.l

```
%{
	int count = 0;
%}
digit [0-9]
letter [a-zA-Z]
eof [;]
operator [+|-|=|/|*|%]
%%
bool|int|float|char { printf("\n<keyword , %s >",yytext);}
{letter}({letter}|{digit})* { printf("< id %d , %s >",++count,yytext);}
{eof} { printf("\n punctuation %s",yytext);}
{operator} { printf("\n<opertor , %s >",yytext);}
{digit}({digit})* { printf("\n<number , %s >",yytext);}
%%
int yywrap()
{return 1;}
int main(int argc, char **argv)
{

	FILE *f1;
	f1 = fopen("sample.c","r");
	yyin = f1;
	yylex();
	printf("Total number of identifier : %d",count);
	return 0;

}
```

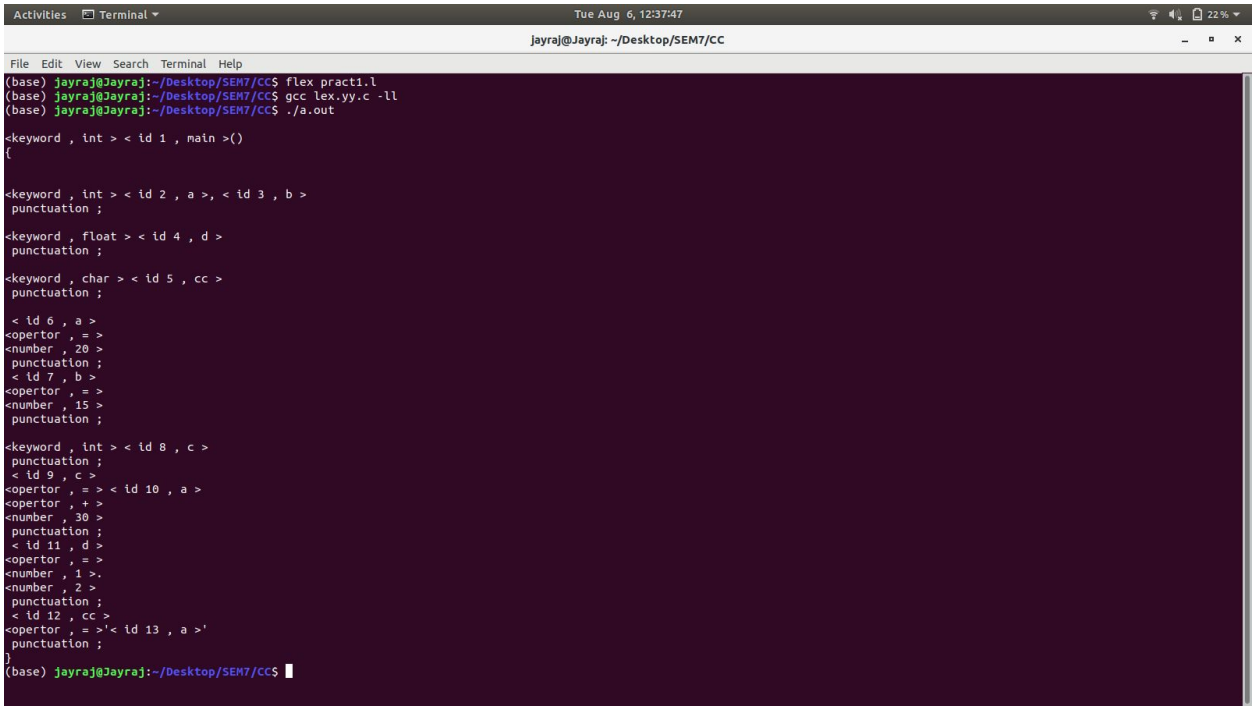
Input:

```
int main()
{

 int a, b;
 float d;
 char cc;

 a = 20;
 b = 15;
 int c;
 c = a + 30;
 d=1.2;
 cc='a';
}
```

Output:



Analysis:

Here the syntaxes that include operators, data types, numbers, etc. have been identified from the input c program. The corresponding identifier/operand/keyword is printed in a structured format, a screenshot of which has been attached here. Regular expressions were used for this purpose.