2nd | 6 | 2020
Thursday

Week 3

1. Write a lex program to find the token and its count from i/p:
(a) Keywords (b) Relop (c) Lower Case letter (d) Upper Case letters (e) Special Characters
(f) Strings within quotes

```
%{
#include<stdio.h>
#include<string.h>
#include<stdlib.h>
int a=0, b=0, c=0, d=0, e=0, f=0;
%}
relop [<|>|=]+
lowcase [a-z]
upcase [A-Z]
letter [a-zA-Z]
Spl [$|%|#|!|@|^|&|*|(|)]
Str [\"]{letter}3+[\"]
%%
"int"|"float"|"char"|"for"|"while"|"do"|"continue"|"break"|"if"|
"then"|"else"|"switch"|"real" {printf("Keyword %s is encountered\n",
yytext); a++;}
{relop} {printf("Relational operator %s is encountered\n", yytext);
b++;}
{lowcase} {printf("Lowercase letter %s encountered\n", yytext);
c++;}
{upcase} {printf("Uppercase letter %s encountered\n", yytext); d++;}
{Spl} {printf("Special character %s is encountered\n", yytext); e++;}
{Str} {printf("String %s is encountered\n", yytext); f++;}
```

```
in Sprintf("\nKeywords = %d \t relop = %d \t
            LowerCase = %d \t UpperCase = %d \t
            SpecialChar = %d \t Strings = %d \n", a, b, c, d, e);
```

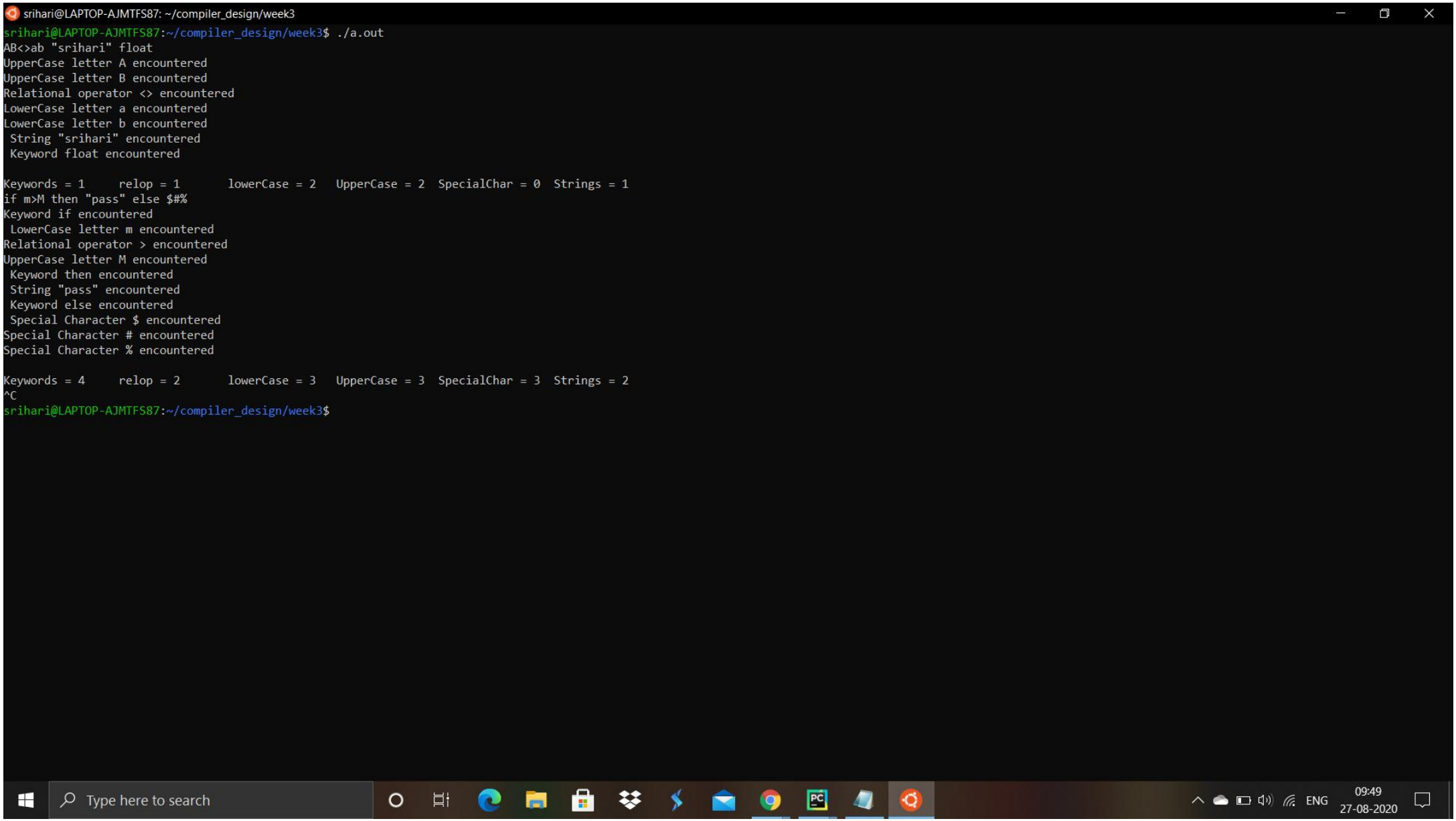
```
%{
int main()
{
    yylex();
    return 0;
}
%}
```

OUTPUT

```
AB<>ab "sihani" float
Uppercase letter A encountered
Uppercase letter B encountered
Relational operator <> encountered
Lowercase letter a encountered
Lowercase letter b encountered
String "sihani" encountered
Keyword float encountered

Keywords = 1   relop = 1   LowerCase = 2   UpperCase = 2   SpecialChar = 0
Strings = 1
```

srihari@LAPTOP-AJMTFS87: ~/compiler_design/week3

```
srihari@LAPTOP-AJMTFS87:~/compiler_design/week3$ cat q1.l
%{
#include<stdio.h>
#include<string.h>
#include<stdlib.h>
int a=0,b=0,c=0,d=0,e=0,f=0;
%}

relop [<|>|=]+
lowcase [a-z]
upcase [A-Z]
letter [a-zA-Z]
spl [$|%|#|!|@|^|&|*|(|)]
str [\"]{letter}+[\"]

%%
"int"|"float"|"char"|"for"|"while"|"do"|"continue"|"break"|"if"|"then"|"else"|"switch"|"real" {printf("Keyword %s encountered\n",yytext);a++;}
{relop} {printf("Relational operator %s encountered\n",yytext);b++;}
{lowcase} {printf("LowerCase letter %s encountered\n",yytext);c++;}
{upcase} {printf("UpperCase letter %s encountered\n",yytext);d++;}
{spl} {printf("Special Character %s encountered\n",yytext);e++;}
{str} {printf("String %s encountered\n",yytext);f++;}
\n {printf("\nKeywords = %d\t relop = %d\t lowerCase = %d\t UpperCase = %d\tSpecialChar = %d\t Strings = %d\n",a,b,c,d,e,f);}
%%

int main(){
//      printf("Enter !!! to stop");
        yylex();
        return 0;
}
srihari@LAPTOP-AJMTFS87:~/compiler_design/week3$
```


srihari@LAPTOP-AJMTFS87: ~/compiler_design/week3
srihari@LAPTOP-AJMTFS87:~/compiler_design/week3$ ./a.out
AB<>ab "srihari" float
UpperCase letter A encountered
UpperCase letter B encountered
Relational operator <> encountered
LowerCase letter a encountered
LowerCase letter b encountered
String "srihari" encountered
Keyword float encountered
Keywords = 1 relop = 1 lowerCase = 2 UpperCase = 2 SpecialChar = 0 Strings = 1
if m>M then "pass" else $#%
Keyword if encountered
LowerCase letter m encountered
Relational operator > encountered
UpperCase letter M encountered
Keyword then encountered
String "pass" encountered
Keyword else encountered
Special Character $ encountered
Special Character # encountered
Special Character % encountered
Keywords = 4 relop = 2 lowerCase = 3 UpperCase = 3 SpecialChar = 3 Strings = 2
^C
srihari@LAPTOP-AJMTFS87:~/compiler_design/week3$
Type here to search
ENG
09:49
27-08-2020

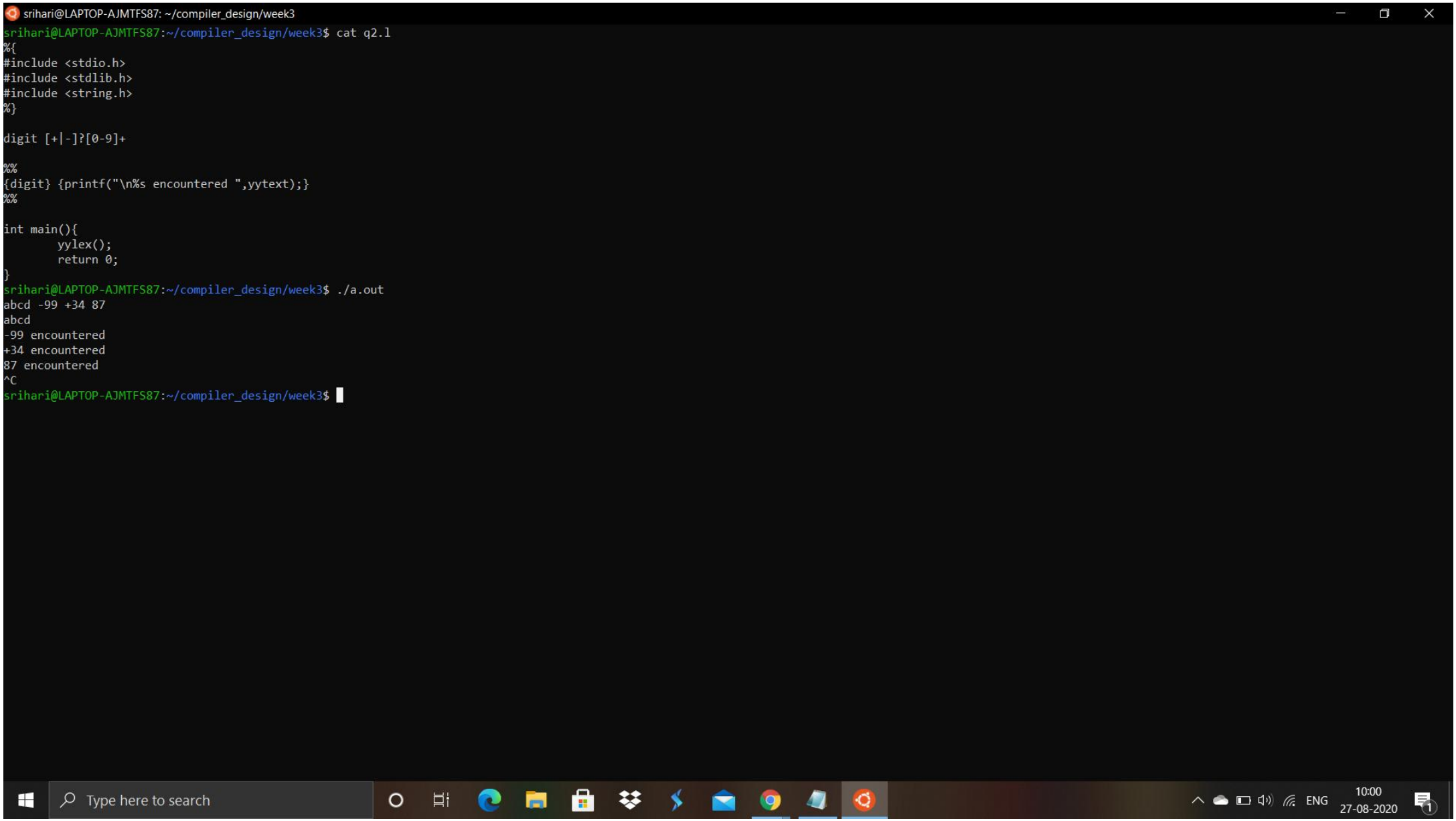
2. Write a lex program to match any string of one or more digits with an optional prefix of + or -.

```
%{
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
%}
digit [+|-]?[0-9]+
```

```
%%
{digit}  { printf("\n%s encountered", yytext); }
%%
int main(){
    yylex();
    return 0;
}
```

OUTPUT

```
abcd  -99  +34  87
abcd
-99 encountered
+34 encountered
87 encountered.
```

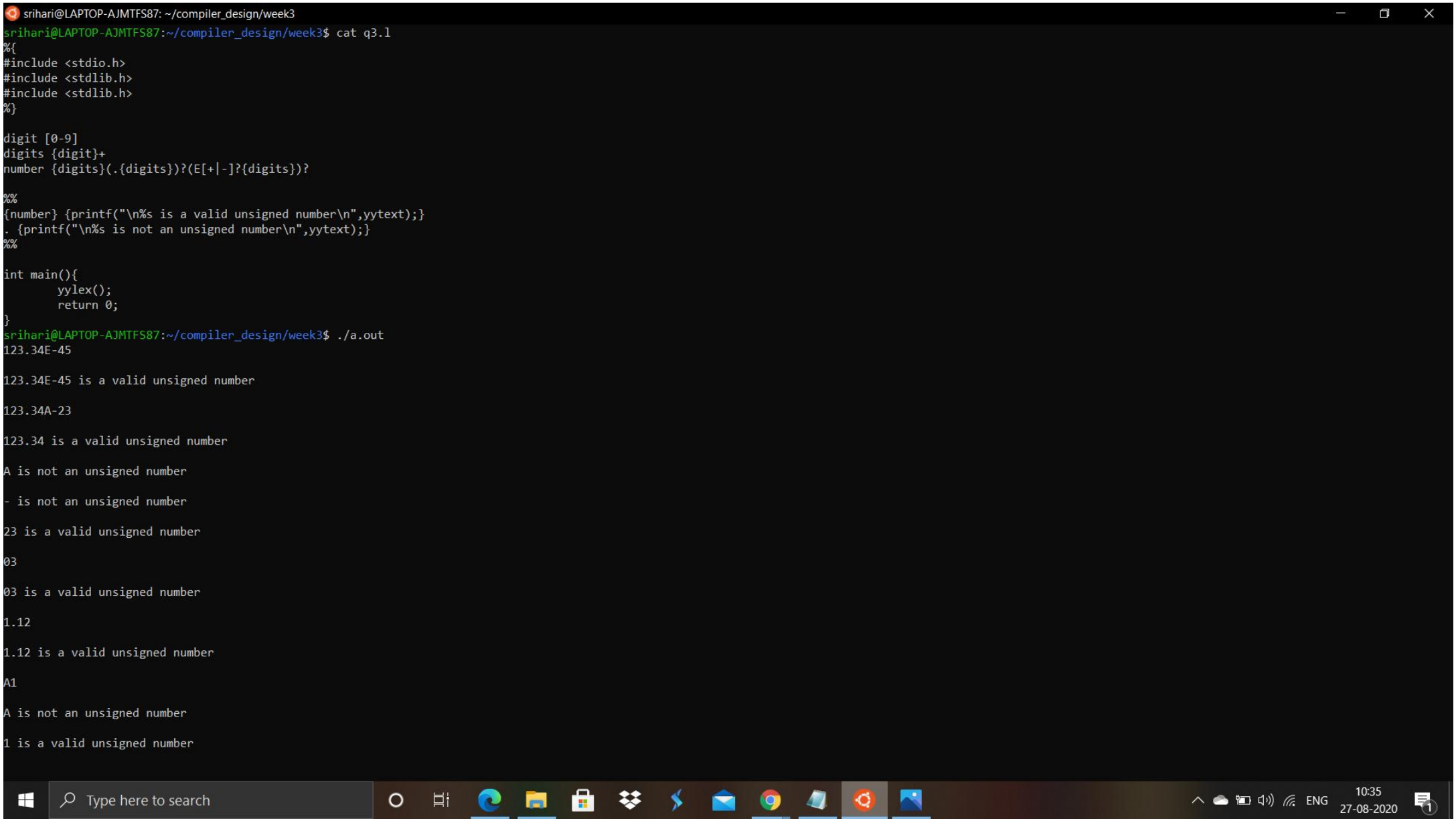

srihari@LAPTOP-AJMTFS87: ~/compiler_design/week3
srihari@LAPTOP-AJMTFS87:~/compiler_design/week3$ cat q2.l
%{
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
%}
digit [+|-]?[0-9]+
%%
{digit} {printf("\n%s encountered ",yytext);}
%%
int main(){
yylex();
return 0;
}
srihari@LAPTOP-AJMTFS87:~/compiler_design/week3$ ./a.out
abcd -99 +34 87
abcd
-99 encountered
+34 encountered
87 encountered
^C
srihari@LAPTOP-AJMTFS87:~/compiler_design/week3$
Type here to search
ENG
10:00
27-08-2020

3. Regular expression for unsigned number. To write a lex program to identify whether a given number is unsigned or not.

```
%{
#include<stdio.h>
#include<string.h>
#include<stdlib.h>
%}
digit [0-9]
digits {digit}+
number {digits}(.{digits})?(E[+|-]?{digits})?

%%
{number} {printf("\n%s is a valid unsigned number\n",yytext);}
. {printf("\n%s is not an unsigned number\n",yytext);}
%%
int main(){
        yylex();
        return 0;
}
```

OUTPUT

```
123.34E-45
123.34E-45 is a valid unsigned number

23
23 is a valid unsigned number

A
A is not an unsigned number.
```


srihari@LAPTOP-AJMTFS87: ~/compiler_design/week3
srihari@LAPTOP-AJMTFS87:~/compiler_design/week3$ cat q3.l
%{
#include <stdio.h>
#include <stdlib.h>
#include <stdlib.h>
%}
digit [0-9]
digits {digit}+
number {digits}(.{digits})?(E[+|-]?{digits})?
%%
{number} {printf("\n%s is a valid unsigned number\n",yytext);}
. {printf("\n%s is not an unsigned number\n",yytext);}
%%
int main(){
yylex();
return 0;
}
srihari@LAPTOP-AJMTFS87:~/compiler_design/week3$ ./a.out
123.34E-45
123.34E-45 is a valid unsigned number
123.34A-23
123.34 is a valid unsigned number
A is not an unsigned number
- is not an unsigned number
23 is a valid unsigned number
03
03 is a valid unsigned number
1.12
1.12 is a valid unsigned number
A1
A is not an unsigned number
1 is a valid unsigned number
Type here to search
ENG
10:35
27-08-2020