GANPAT UNIVERSITY

U.V. PATEL COLLEGE OF ENGINEERING

PRACTICAL File

Student Name:	**Sureja Dharmay Dineshchandra**
Enrollment No:	**17012011056**
Subject Name:	**Theory Of Computation**
Subject Code:	**2CE601**
Semester:	**B.Tech 6th Sem**
Branch:	**Computer Engineering**

INDEX

PRACTICALS

Sr. No.	PRACTICAL Aim	Date of Start	Date of Complete	Page Start	Page End	Sign	Grade / Marks
1	Introduction of “VI” Editor with Commands and lex			**1**	**4**		
2	Write a separate Lex programs			**5**	**33**		
3	Write a Lex program to recognize identifiers in C			**34**	**37**		
4	Write a lex program to count number of words, characters and lines from user input			**38**	**43**		
5	Write Lex program for given questions.			**44**	**48**		
6	Write a program to identify the word and change the case of each character of word (i.e if character in word is in lowercase then convert it to uppercase and if it is uppercase then convert to lowercase)			**49**	**53**		
7	Given NFA for (11+110)*0 . Write a program to find δ* (q0, 110) and δ* (q0, 11)			**54**	**57**		

CERTIFICATE

GANPAT UNIVERSITY
॥ विद्यया समाजोत्कर्षः ॥

U V PATEL COLLEGE OF ENGINEERING

T O   W H O M   S O   E V E R   I T   M A Y   C O N C E R N

This is to certify that Mr. **Sureja Dharmay Dineshchandra** a student of **B.Tech Semester VI Computer Engineering** with Enrollment No. **17012011056** of **6th CE** Class has satisfactorily completed the course and submitted the term work of **Theory of Computation(2CE601)** satisfactorily for the fulfillment of semester end examination.

Date of Submission:

Subject Teacher

Head of the Department

PRACTICAL-1

1. Introduction of "VI" Editor with Commands.

There are many ways to edit files in UNIX and one of the best ways is using screen-oriented text editor vi. It's usually available on all the flavors of UNIX system. Its implementations are very similar across the board. It requires very few resources. It is more user friendly than any other editors.
Some Commands of VI Editor are as follows.

- vi filename: Creates a new file if it already does not exist, otherwise opens existing file.
- vi -R filename: Opens an existing file in read only mode.
- TOUCH: Use to create empty file or to update timestamp of an existing file.
- LS: List the file is current directory in alphanumeric order. It also display directories' name in current directory.
- MDIR: Removes directory.
- CLEAR: Clear the contents of the screen.
- CHMOD: Set/Change the permission of one or more file for user.
- WHO:Gives list of all users who are currently logged on to the system.
- CD:Used to change the current working directory.
- MKDIR: Use to create new directories.

2. Introduction of Lex.

Lex is a tool that reads a stream of input from somewhere and breaks it up into its component pieces. Each component piece is a token. A token might be a keyword or a number or a string or punctuation. In this context a token is something that cannot be broken down into smaller pieces. Either it's a keyword or it's something else. If it's a string it's a string and we are not concerned with the machine representation of a string. Likewise with numbers and punctuation.

The first section contains general c code declarations and lex directives and is delimited from the second section by a %% line.

The second section contains regular expressions of a lex file. The second section is delimited from the third section by a %% line.

The third section contains raw c code which is copied verbatim to the output file.

And a lex file looks like this

```
%{
C declarations
%}
lex declarations
%%
Regular expressions
%%
Additional C code
```

The program Lex generates a so called `Lexer'. This is a function that takes a stream of characters as its input, and whenever it sees a group of characters that match a key, takes a certain action. A very simple example:

```
%%
/* match everything except newline */
. ECHO;
/* match newline */
\n ECHO;
%%
int yywrap(void) {
return 1;
}
int main(void) {
yylex();
return 0;
}
```

Two patterns have been specified in the rules section. Each pattern must begin in column one. This is followed by whitespace (space, tab or newline) and an optional action associated with the pattern. The action may be a single C statement, or multiple C statements, enclosed in braces. Anything not starting in column one is copied verbatim to the generated C file. We may take advantage of this behavior to specify comments in our lex file. In this example there are two patterns, "." and "\n", with an ECHO action associated for each pattern. Several macros and variables are predefined by lex. ECHO is a macro that writes code matched by the pattern. This is the default action for any unmatched strings.

```
%{
#include <stdio.h>
%}
%%
stopprintf("Stop command received\n");
startprintf("Start command received\n");
%%
```

The first section, in between the %{ and %} pair is included directly in the output program. We need this, because we use printf later on, which is defined in stdio.h. Sections are separated using '%%', so the first line of the second section starts with the 'stop' key. Whenever the 'stop' key is encountered in the input, the rest of the line (a printf() call) is executed. Besides 'stop', we've also defined 'start', which otherwise does mostly the same. We terminate the code section with '%%' again.

Matching patterns of text

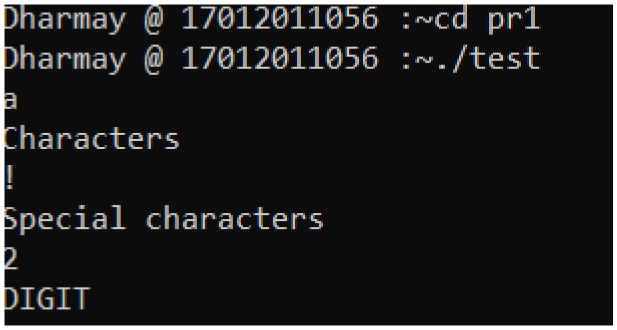
Expression	Matches
.	any single character
*	zero or more of the previous expression
.*	zero or more arbitrary characters
\<	beginning of a word
\>	end of a word
\	quote a special character
*	the character ``*"
^	beginning of a line
$	end of a line
[*set*]	one character from a set of characters
[XYZ]	one of the characters ``X", ``Y", or ``Z"
[[:upper:]][[:lower:]]*	one uppercase character followed by any number of lowercase characters
[^*set*]	one character not from a set of characters
[^XYZ[:digit:]]	any character except ``X", ``Y", ``Z", or a numeric digit

Ex:Check given input is digit or character or special character in lex program.

Code:

```
%{
  #include <stdio.h>
%}
%%
[A-Za-z]+ printf("Characters");
[0-9]+ printf("DIGIT");
.* printf("Special characters");

%%
int yywrap(void)
{
 return 1;
}
int main(void)
{
   yylex();
}
```

OutPut:



```
Dharmay @ 17012011056 :~cd pr1
Dharmay @ 17012011056 :~./test
a
Characters
!
Special characters
2
DIGIT
```

PRACTICAL-2

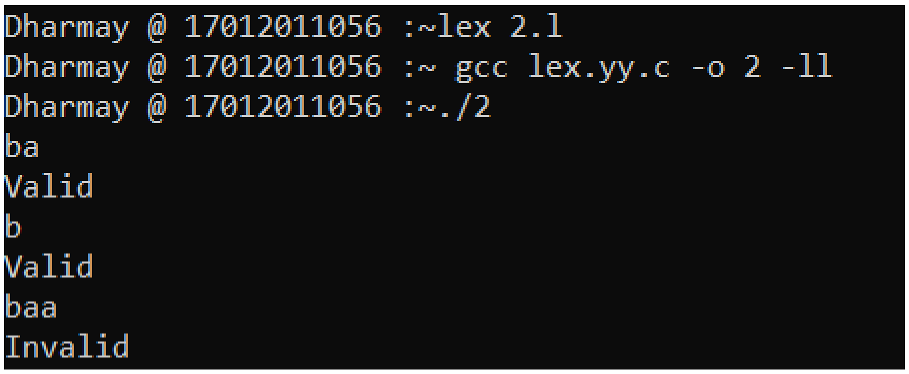
1. **Write a separate Lex programs for following.**

   a. **Write RE that accept zero or one(at most one) occurrence of 'a'.**

   Code:

```
  %{
 #include<stdio.h>
%}
%%
ba? printf("Valid");
.* printf("Invalid");
%%
int yywrap(void)
{
 return 1;
}
int main(void)
{
 yylex();
}
```

   Output:



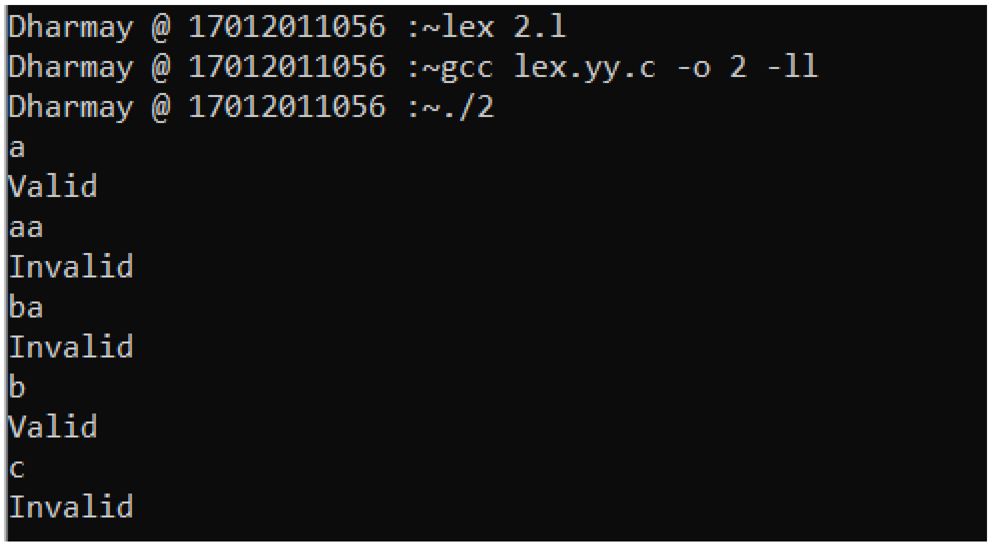
```
Dharmay @ 17012011056 :~lex 2.l
Dharmay @ 17012011056 :~ gcc lex.yy.c -o 2 -ll
Dharmay @ 17012011056 :~./2
ba
Valid
b
Valid
baa
Invalid
```

   b. **Write RE that accept either 'a' or 'b'.**

   Code:

```
 %{
  #include<stdio.h>
%}
%%
a|b printf("Valid");
.* printf("Invalid");
%%
int yywrap(void)
{
 return 1;
}
int main(void)
{
  yylex();
}
```

Output:



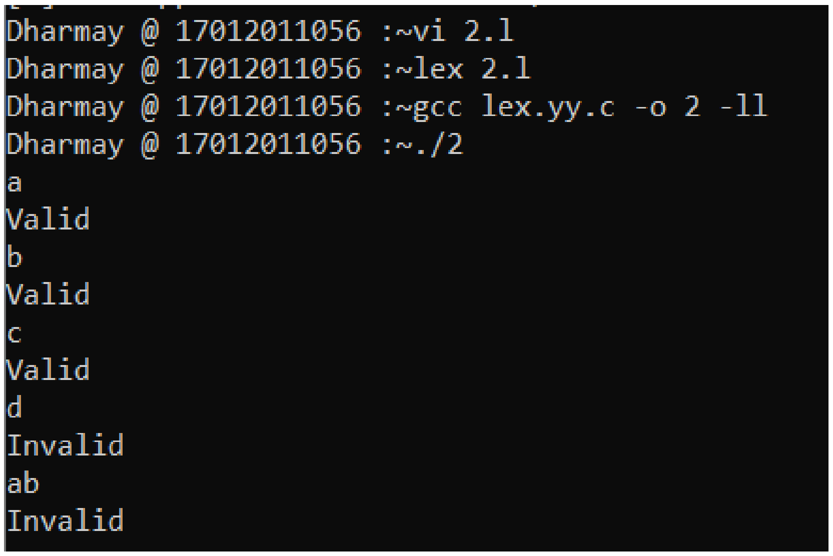
```
Dharmay @ 17012011056 :~lex 2.l
Dharmay @ 17012011056 :~gcc lex.yy.c -o 2 -ll
Dharmay @ 17012011056 :~./2
a
Valid
aa
Invalid
ba
Invalid
b
Valid
c
Invalid
```

c. **Write RE that accept either ‘a’ or ‘b’ or ‘c’ without using |.**

Code:

```
 %{
  #include<stdio.h>
%}
%%
[a,b,c] printf("Valid");
.* printf("Invalid");
%%
int yywrap(void)
{
 return 1;
}
int main(void)
{
  yylex();
}
```

__Output:__



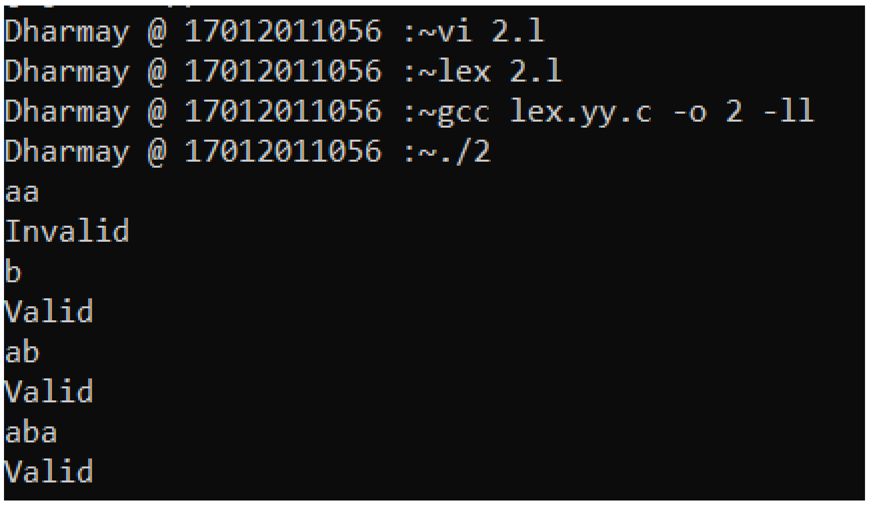
```
Dharmay @ 17012011056 :~vi 2.l
Dharmay @ 17012011056 :~lex 2.l
Dharmay @ 17012011056 :~gcc lex.yy.c -o 2 -ll
Dharmay @ 17012011056 :~./2
a
Valid
b
Valid
c
Valid
d
Invalid
ab
Invalid
```

d. Write RE that accept that accept zero or more occurrences of ‘a’ and single occurrences of ‘b’.

__Code:__

```
%{
  #include<stdio.h>
%}
%%
a*ba* printf("Valid");
.* printf("Invalid");
%%
int yywrap(void)
{
 return 1;
}
int main(void)
{
  yylex();
}
```

Output:



```
Dharmay @ 17012011056 :~vi 2.l
Dharmay @ 17012011056 :~lex 2.l
Dharmay @ 17012011056 :~gcc lex.yy.c -o 2 -ll
Dharmay @ 17012011056 :~./2
aa
Invalid
b
Valid
ab
Valid
aba
Valid
```

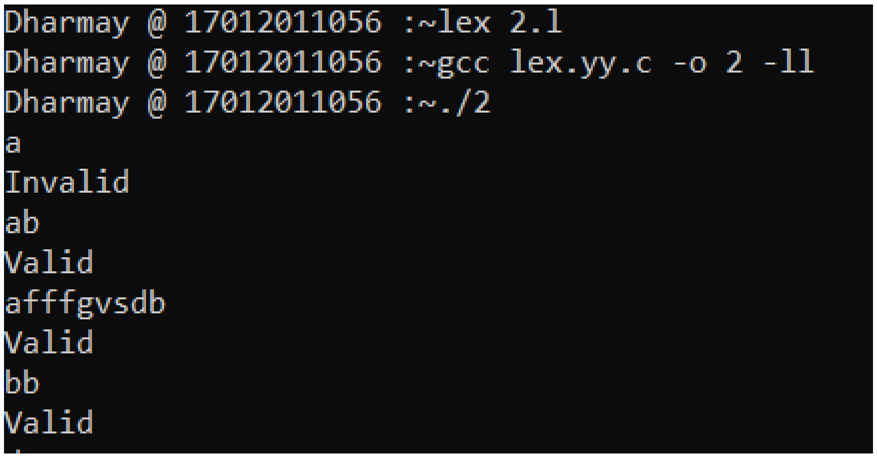
e. **Write RE that accepts all the strings which ends with 'b'.**

Code:

```
%{
  #include<stdio.h>
%}
%%
.*b printf("Valid");
```

```
.* printf("Invalid");
%%
int yywrap(void)
{
 return 1;
}
int main(void)
{
  yylex();
}
```

Output:



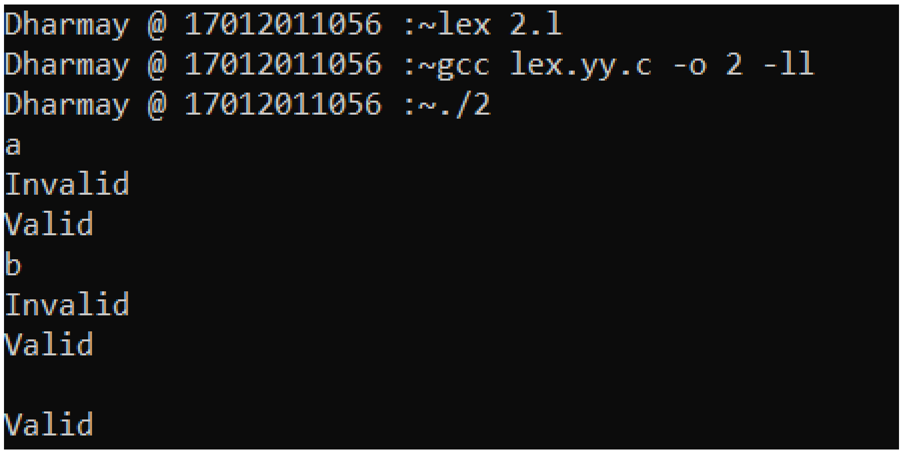
```
Dharmay @ 17012011056 :~lex 2.l
Dharmay @ 17012011056 :~gcc lex.yy.c -o 2 -ll
Dharmay @ 17012011056 :~./2
a
Invalid
ab
Valid
afffgvsdb
Valid
bb
Valid
```

f. **Write RE for new line.**

Code:

```
%{
  #include<stdio.h>
%}
%%
[\n] printf("Valid\n");
.* printf("Invalid\n");
%%
int yywrap(void)
{
 return 1;
}
int main(void)
{
  yylex();
}
```

Output:



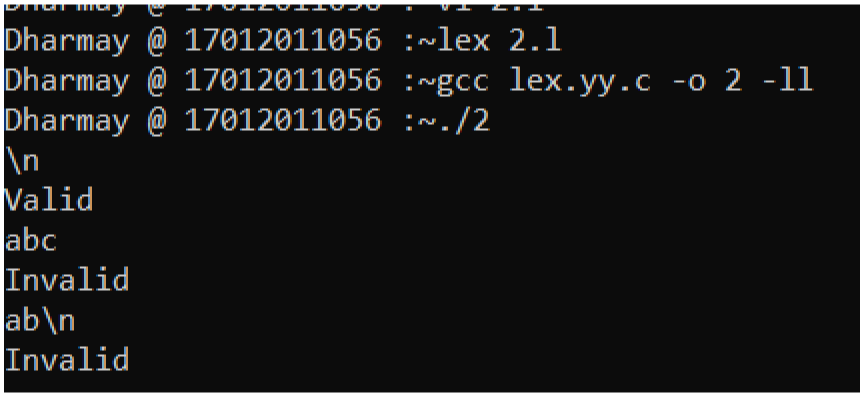
```
Dharmay @ 17012011056 :~lex 2.l
Dharmay @ 17012011056 :~gcc lex.yy.c -o 2 -ll
Dharmay @ 17012011056 :~./2
a
Invalid
Valid
b
Invalid
Valid

Valid
```

g. **Write RE that accepts '\n'.**

Code:

```
%{
  #include<stdio.h>
%}
%%
\\n printf("Valid");
.* printf("Invalid");
%%
int yywrap(void)
{
 return 1;
}
int main(void)
{
  yylex();
}
```

Output:



```
Dharmay @ 17012011056 :~lex 2.l
Dharmay @ 17012011056 :~gcc lex.yy.c -o 2 -ll
Dharmay @ 17012011056 :~./2
\n
Valid
abc
Invalid
ab\n
Invalid
```

h. **Write a string that accepted by the given lex program and justify the output.**

1.	2.

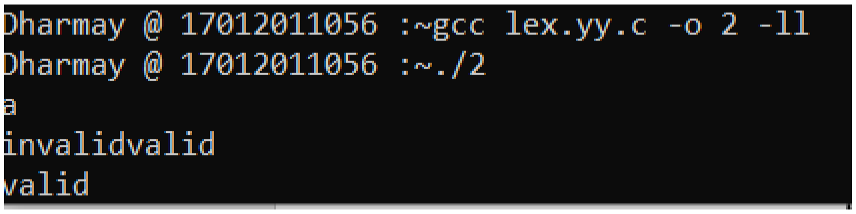
%{
#include<stdio.h>
%}
 	
%%
[\n] printf("valid");
.* printf("invalid");
%%
int yywrap()
{
return 1;
}
int main()
{
yylex();
return 0;
}
	%{
	#include<stdio.h>
	%}
	%%
	[\\n] printf("valid");
	.* printf("invalid");
	%%
	int yywrap(void)
	{
	return 1;
	}
	int main()
	{
	yylex();
	return 0;
	}

Code1:

```
%{
    #include<stdio.h>
    %}
    %%
```

```
[\n] printf("valid");
.* printf("invalid");
%%
int yywrap()
{
        return 1;
}
int main()
{
        yylex();
        return 0;
}
```

Output1:

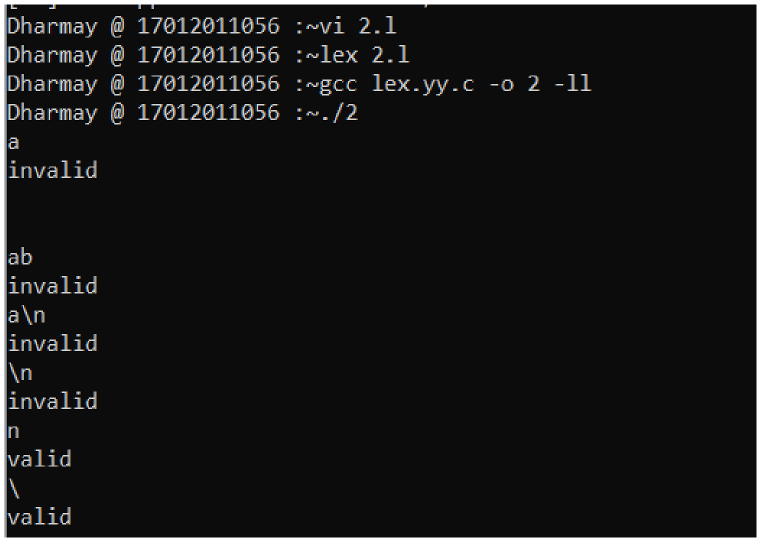


```
Dharmay @ 17012011056 :~gcc lex.yy.c -o 2 -ll
Dharmay @ 17012011056 :~./2
a
invalidvalid
valid
```

Code2:

```
%{
    #include<stdio.h>
    %}
    %%
    [\\n]  printf("valid");
    .* printf("invalid");
    %%
    int yywrap(void)
    {
    return 1;
    }
    int main()
    {
    yylex();
    return 0;
    }
```

Output2:



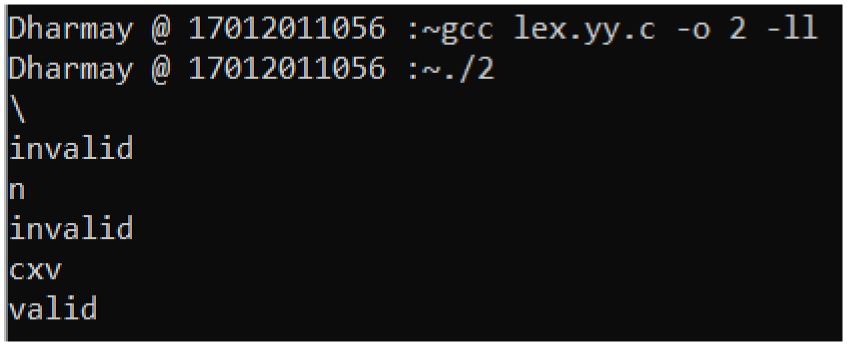
```
Dharmay @ 17012011056 :~vi 2.l
Dharmay @ 17012011056 :~lex 2.l
Dharmay @ 17012011056 :~gcc lex.yy.c -o 2 -ll
Dharmay @ 17012011056 :~./2
a
invalid


ab
invalid
a\n
invalid
\n
invalid
n
valid
\
valid
```

i. **Write a RE that accepts any character except '\' and 'n'.**

<u>Code:</u>

```
%{
#include<stdio.h>
%}
%%
[\\n]  printf("invalid");
.* printf("valid");
%%
int yywrap(void)
{
return 1;
}
int main()
{
yylex();
return 0;
}
```

<u>Output:</u>



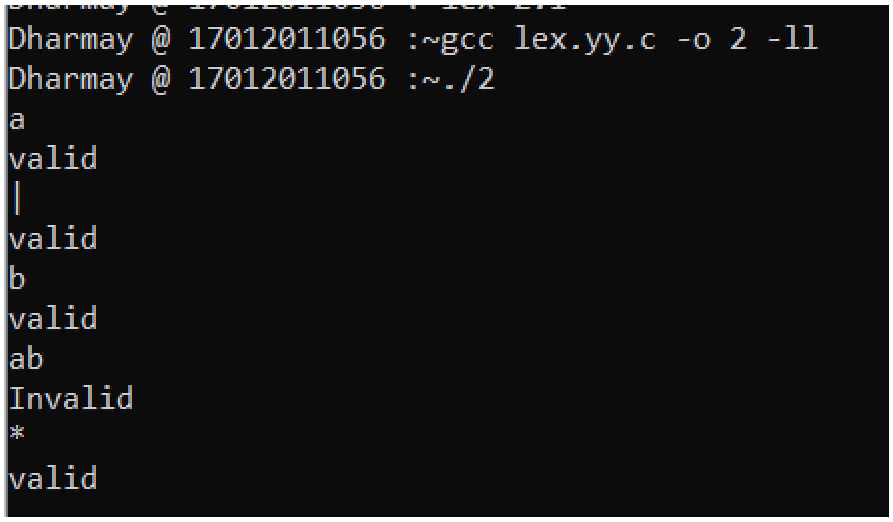
```
Dharmay @ 17012011056 :~gcc lex.yy.c -o 2 -ll
Dharmay @ 17012011056 :~./2
\
invalid
n
invalid
cxv
valid
```

j. Write all the strings which are accepted by [a|b|c*].

Code:

```
%{
#include<stdio.h>
%}
%%
[a|b|c*]  printf("valid");
.* printf("Invalid");
%%
int yywrap(void)
{
return 1;
}
int main()
{
yylex();
return 0;
}
```

Output:



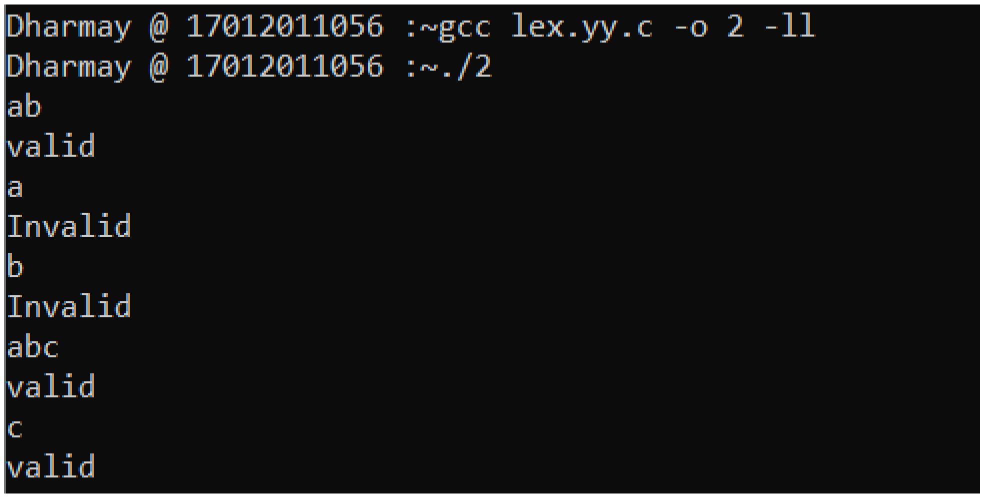
```
Dharmay @ 17012011056 :~gcc lex.yy.c -o 2 -ll
Dharmay @ 17012011056 :~./2
a
valid
|
valid
b
valid
ab
Invalid
*
valid
```

k. Write a RE that accept any character except ‘a’ and ‘b’.

Code:

```
%{
#include<stdio.h>
%}
%%
a|b printf("Invalid");
.* printf("valid");
%%
int yywrap(void)
{
return 1;
}
int main()
{
yylex();
return 0;
}
```

Output:



```
Dharmay @ 17012011056 :~gcc lex.yy.c -o 2 -ll
Dharmay @ 17012011056 :~./2
ab
valid
a
Invalid
b
Invalid
abc
valid
c
valid
```

l. **Write more than one RE that accepts string ‘abc’.**

<u>Code(1):</u>

```
%{
#include<stdio.h>
%}
%%
abc printf("valid");
.* printf("Invalid");
%%
int yywrap(void)
{
return 1;
}
int main()
{
yylex();
return 0;
}
```

<u>Code(2):</u>

```
%{
#include<stdio.h>
%}
%%
abc printf("valid");
.* printf("Invalid");
%%
int yywrap(void)
{
return 1;
}
int main()
{
yylex();
return 0;
}
```

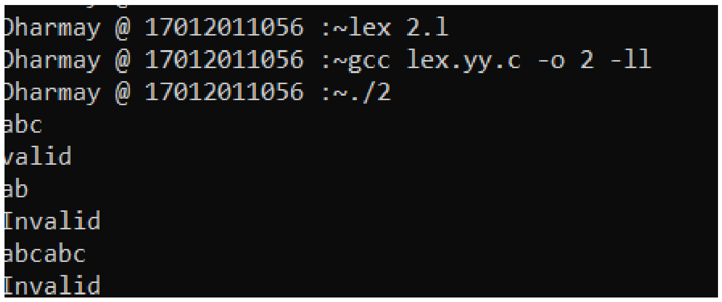
<u>Code(3):</u>

```
%{
#include<stdio.h>
%}
%%
(abc) printf("valid");
.* printf("Invalid");
%%
int yywrap(void)
{
return 1;
}
int main()
{
yylex();
return 0;
}
```

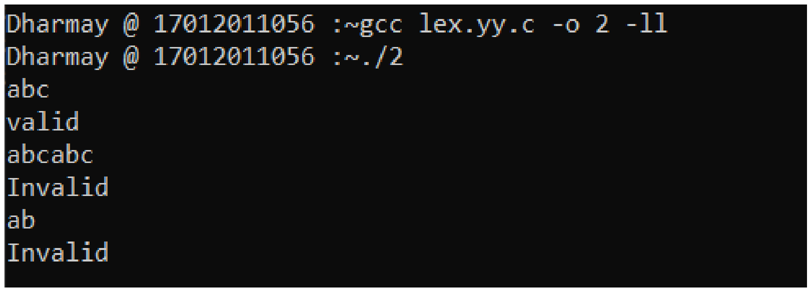
<u>Code(4):</u>

```
%{
#include<stdio.h>
%}
%%
[a][b][c] printf("valid");
.* printf("Invalid");
%%
int yywrap(void)
{
return 1;
}
int main()
{
yylex();
return 0;
}
```

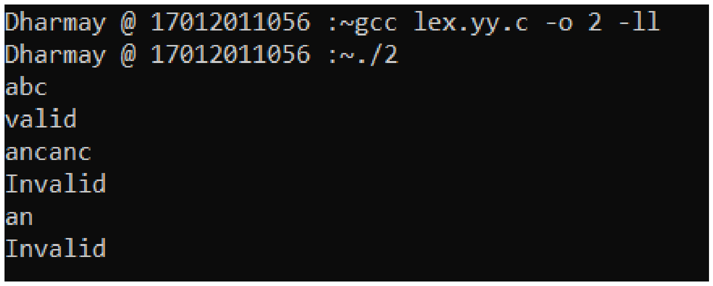
Output:



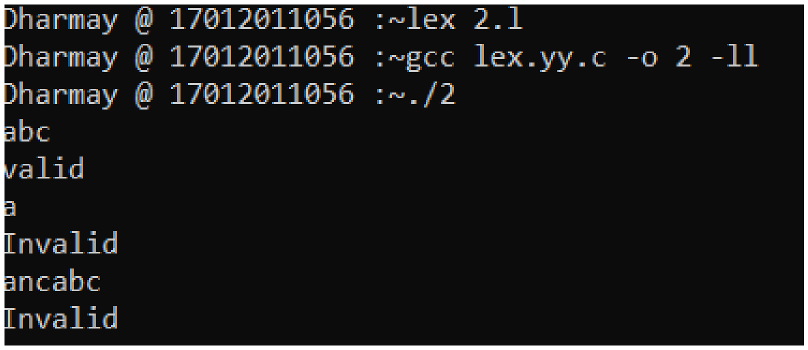
```
Dharmay @ 17012011056 :~lex 2.l
Dharmay @ 17012011056 :~gcc lex.yy.c -o 2 -ll
Dharmay @ 17012011056 :~./2
abc
valid
ab
Invalid
abcabc
Invalid
```



```
Dharmay @ 17012011056 :~gcc lex.yy.c -o 2 -ll
Dharmay @ 17012011056 :~./2
abc
valid
abcabc
Invalid
ab
Invalid
```



```
Dharmay @ 17012011056 :~gcc lex.yy.c -o 2 -ll
Dharmay @ 17012011056 :~./2
abc
valid
ancanc
Invalid
an
Invalid
```



```
Dharmay @ 17012011056 :~lex 2.l
Dharmay @ 17012011056 :~gcc lex.yy.c -o 2 -ll
Dharmay @ 17012011056 :~./2
abc
valid
a
Invalid
ancabc
Invalid
```

m. Is there any difference between ‘abc’ and “abc”? Justify your answer.

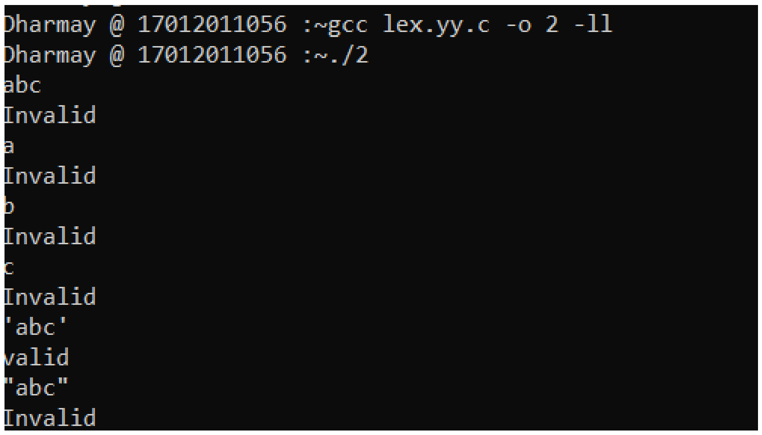
<u>Code(1):</u>

```
%{
#include<stdio.h>
%}
%%
'abc' printf("valid");
.* printf("Invalid");
%%
int yywrap(void)
{
return 1;
}
int main()
{
yylex();
return 0;
}
```

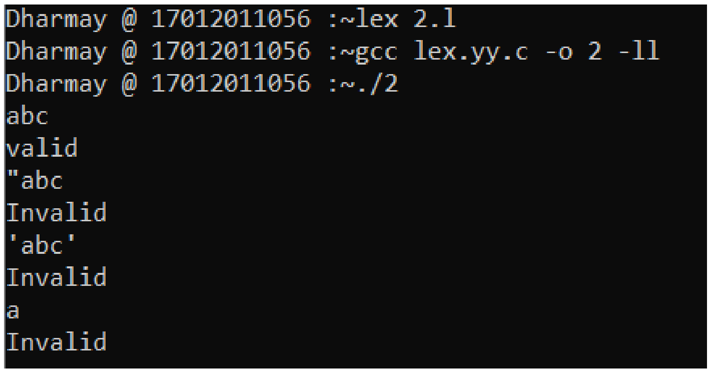
Code(2):

```
%{
#include<stdio.h>
%}
%%
"abc" printf("valid");
.* printf("Invalid");
%%
int yywrap(void)
{
return 1;
}
int main()
{
yylex();
return 0;
}
```

Output:



```
Dharmay @ 17012011056 :~gcc lex.yy.c -o 2 -ll
Dharmay @ 17012011056 :~./2
abc
Invalid
a
Invalid
b
Invalid
c
Invalid
'abc'
valid
"abc"
Invalid
```



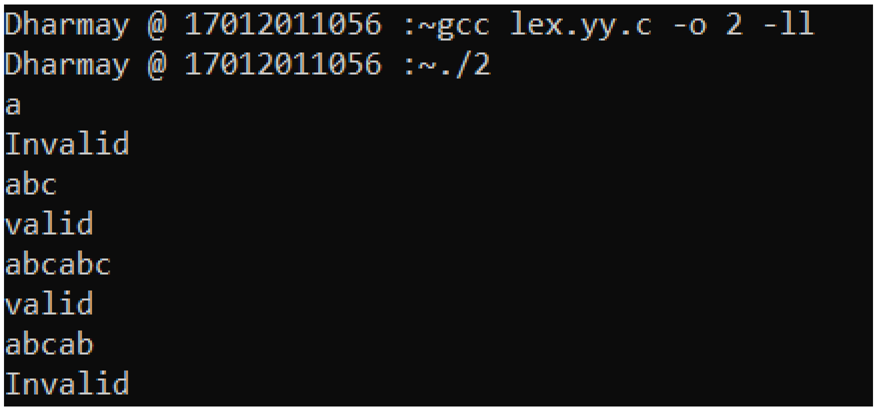
```
Dharmay @ 17012011056 :~lex 2.l
Dharmay @ 17012011056 :~gcc lex.yy.c -o 2 -ll
Dharmay @ 17012011056 :~./2
abc
valid
"abc
Invalid
'abc'
Invalid
a
Invalid
```

n. Which are the strings accepted by (“abc”)*.

Code:

```
%{
#include<stdio.h>
%}
%%
("abc")* printf("valid");
.* printf("Invalid");
%%
int yywrap(void)
{
return 1;
}
int main()
{
yylex();
return 0;
}
```

Output:



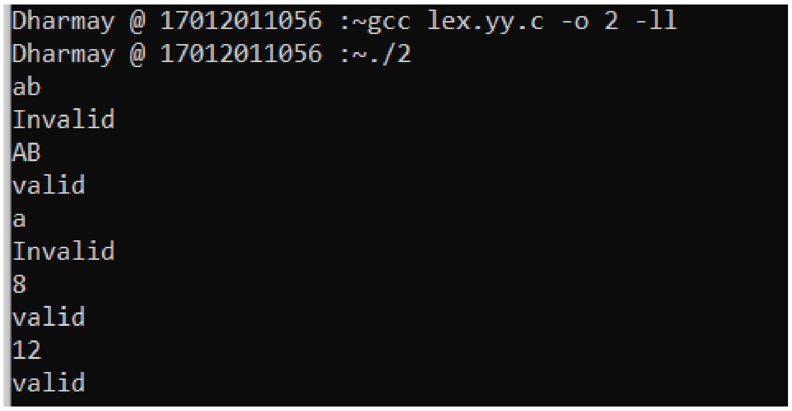
```
Dharmay @ 17012011056 :~gcc lex.yy.c -o 2 -ll
Dharmay @ 17012011056 :~./2
a
Invalid
abc
valid
abcabc
valid
abcab
Invalid
```

o. Write the RE that accepts zero or more occurrences of digit and capital letters.

Code:

```
%{
#include<stdio.h>
%}
%%
[A-Z0-9]* printf("valid");
.* printf("Invalid");
%%
int yywrap(void)
{
return 1;
}
int main()
{
yylex();
return 0;
}
```

Output:



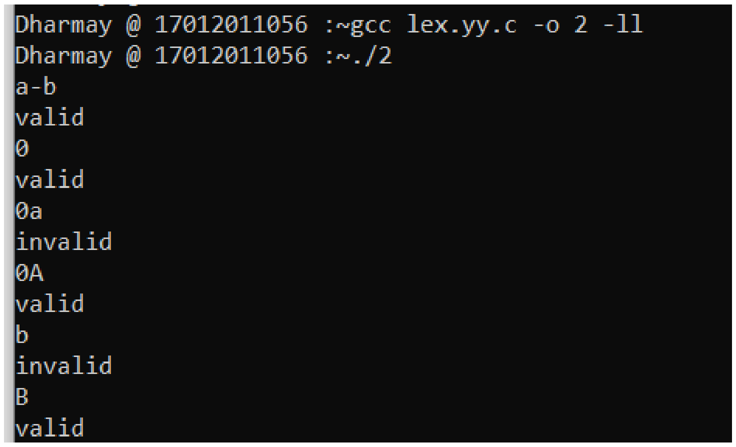
```
Dharmay @ 17012011056 :~gcc lex.yy.c -o 2 -ll
Dharmay @ 17012011056 :~./2
ab
Invalid
AB
valid
a
Invalid
8
valid
12
valid
```

p. Write valid and invalid strings accepted by following regular expressions.

I. (a-b)?[0-9 A-Z]*

```
%{
#include <stdio.h>
%}
%%
(a-b)?[A-Z0-9]* printf( "valid");
.* printf("invalid");
%%
int yywrap(void)
{
return 1;
}
int main()
{
yylex();
return 0;
}
```

Output:



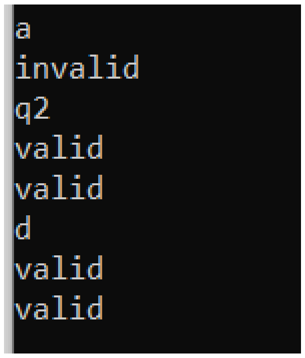
```
Dharmay @ 17012011056 :~gcc lex.yy.c -o 2 -ll
Dharmay @ 17012011056 :~./2
a-b
valid
0
valid
0a
invalid
0A
valid
b
invalid
B
valid
```

II. [^ab][0-9]*

Code:

```
%{
#include <stdio.h>
%}
%%
[^ab][0-9]* printf( "valid\n");
.* printf("invalid\n");
%%
int yywrap(void)
{
  return 1;
}
int main()
{
    yylex();
    return 0;
}
```

Output:



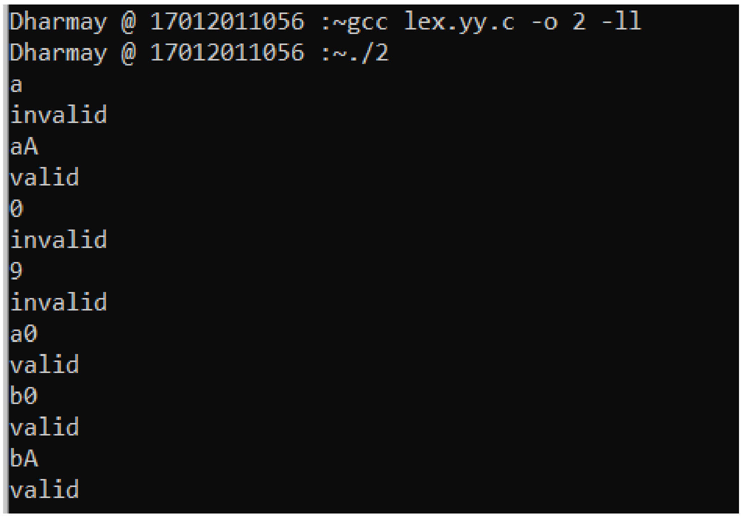
```
a
invalid
q2
valid
valid
d
valid
valid
```

III. ^[ab][0-9 A-Z]+

Code:

```
%{
#include <stdio.h>
%}
%%
^[ab][0-9A-Z]+ printf( "valid");
.* printf("invalid");
%%
int yywrap(void)
{
return 1;
}
int main()
{
   yylex();
   return 0;
}
```

Output:



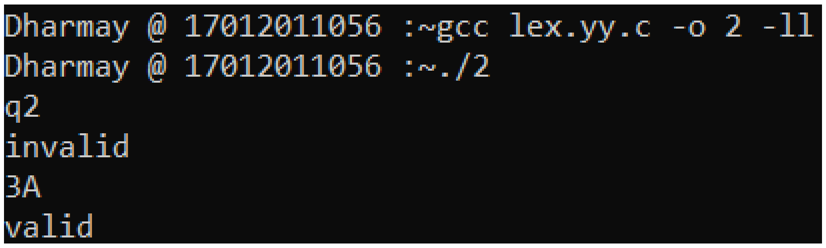
```
Dharmay @ 17012011056 :~gcc lex.yy.c -o 2 -ll
Dharmay @ 17012011056 :~./2
a
invalid
aA
valid
0
invalid
9
invalid
a0
valid
b0
valid
bA
valid
```

IV. [0-9][A-Z]$

Code:

```
%{
#include <stdio.h>
%}
%%
[0-9][A-Z]$ printf( "valid");
.* printf("invalid");
%%
int yywrap(void)
{
return 1;
}
int main()
{
   yylex();
   return 0;
}
```

Output:



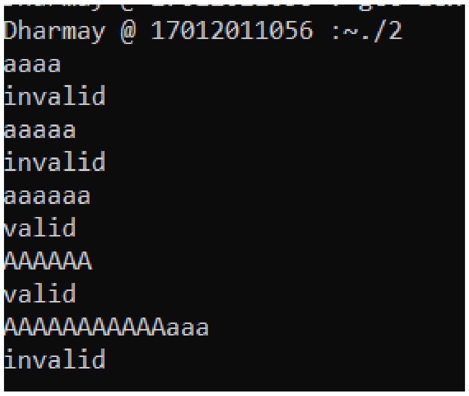
```
Dharmay @ 17012011056 :~gcc lex.yy.c -o 2 -ll
Dharmay @ 17012011056 :~./2
q2
invalid
3A
valid
```

V. [A-Z a-z]{6}

Code:

```
%{
#include <stdio.h>
%}
%%
[A-Za-z]{6} {printf( "valid");}
.* {printf("invalid");}
%%
int yywrap(void)
{
return 1;
}
int main()
{
   yylex();
   return 0;
}
```

Output:



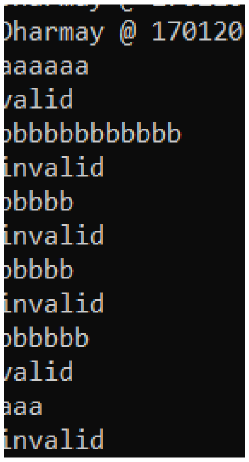
```
Dharmay @ 17012011056 :~./2
aaaa
invalid
aaaaa
invalid
aaaaaa
valid
AAAAAA
valid
AAAAAAAAAAAaaa
invalid
```

VI. [a+b]{6}

Code:

```
%{
#include <stdio.h>
%}
%%
[a+b]{6} printf( "valid");
.* printf("invalid");
%%
int yywrap(void)
{
return 1;
}
int main()
{
   yylex();
   return 0;
}
```

Output:

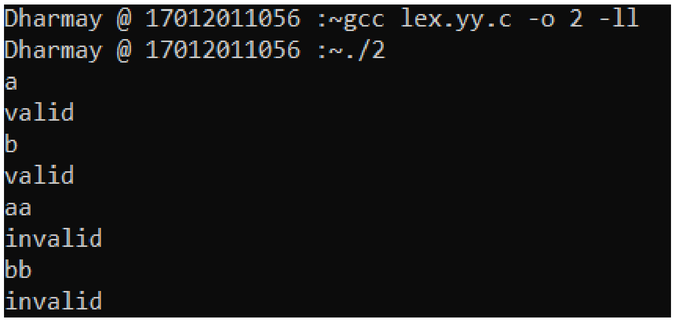

```
Dharmay @ 170120
aaaaaa
valid
bbbbbbbbbbbb
invalid
bbbbb
invalid
bbbbb
invalid
bbbbbb
valid
aaa
invalid
```

VII. [a+b]

Code:

```
%{
#include <stdio.h>
%}
%%
[a+b] printf( "valid");
.* printf("invalid");
%%
int yywrap()
{
return 1;
}
int main()
{
   yylex();
   return 0;
}
```

Output:



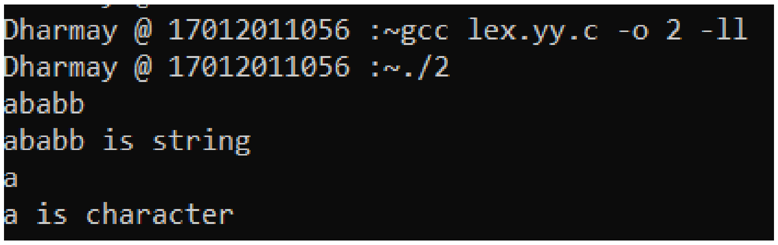
```
Dharmay @ 17012011056 :~gcc lex.yy.c -o 2 -ll
Dharmay @ 17012011056 :~./2
a
valid
b
valid
aa
invalid
bb
invalid
```

2. **Demonstrate the use of lex predefined variables (yytext, yyleng, yyin) with the help of program.**

<u>Code:</u>

```
%{
  #include<stdio.h>
%}
%%
[a-zA-Z] {printf("%s is character",yytext);}
[a-zA-Z]* {printf("%s is string",yytext);}
.* {printf("%s special symbols",yytext);}
%%
        int yywrap()
        {
          return 1;
        }
        int main()
        {
          yylex();
          return 0;
        }
```

<u>Output:</u>



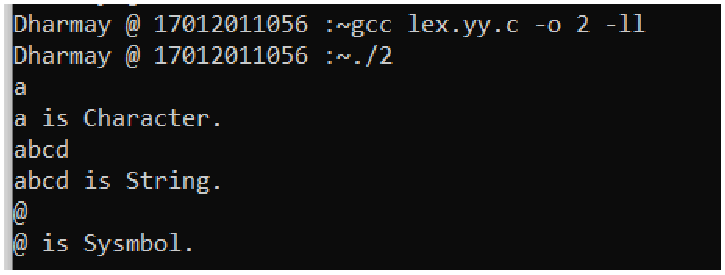
```
Dharmay @ 17012011056 :~gcc lex.yy.c -o 2 -ll
Dharmay @ 17012011056 :~./2
ababb
ababb is string
a
a is character
```

3. **Write a lex program to recognize character, string and special symbols from given input.**

<u>Code:</u>

```
%{
#include <stdio.h>
%}
%%
[a-zA-Z0-9] printf( "%s is Character.",yytext);
. printf("%s is Sysmbol.",yytext);
.* printf("%s is String.",yytext);

%%
int yywrap()
{
 return 1;
}
int main()
{
       yylex();
    return 0;
}
```

<u>Output:</u>



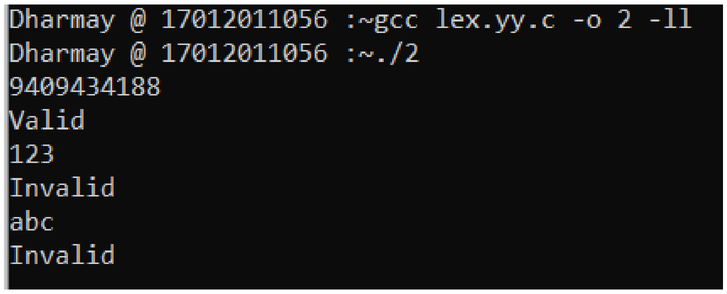
```
Dharmay @ 17012011056 :~gcc lex.yy.c -o 2 -ll
Dharmay @ 17012011056 :~./2
a
a is Character.
abcd
abcd is String.
@
@ is Sysmbol.
```

4. Write a lex program to validate mobile number. (i.e Number having length of 10 is valid)

Code:

```
%{
#include <stdio.h>
%}
%%
[0-9]{10} printf("Valid");
.* printf("Invalid");
%%
int yywrap(void)
{
return 1;
}
int main()
{
   yylex();
   return 0;
}
```

Output:



```
Dharmay @ 17012011056 :~gcc lex.yy.c -o 2 -ll
Dharmay @ 17012011056 :~./2
9409434188
Valid
123
Invalid
abc
Invalid
```

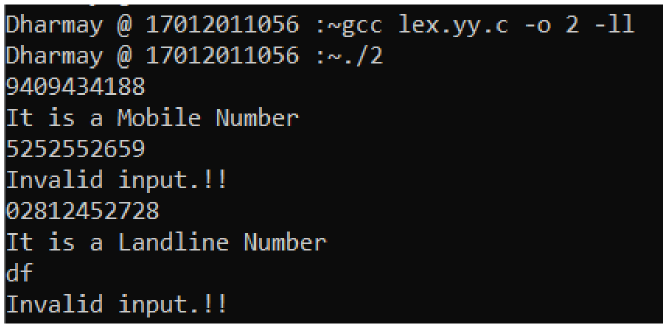
5. Write a lex program to differentiate mobile number and land line number.

Code:

```
%{
#include <stdio.h>
%}
%%
[0-6]+[7-9]+[0-9]{8} printf("It is a Landline Number");
[6-9][0-9]{9} printf("It is a Mobile Number");
```

```
.* printf("Invalid input.!!");
%%
int yywrap(void)
{
return 1;
}
int main()
{
   yylex();
   return 0;
}
```

Output:

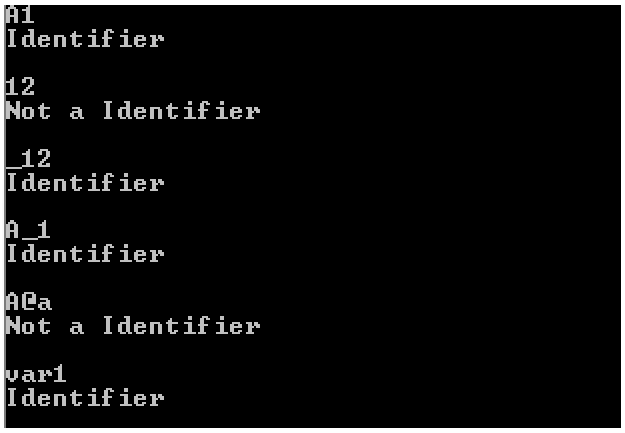


```
Dharmay @ 17012011056 :~gcc lex.yy.c -o 2 -ll
Dharmay @ 17012011056 :~./2
9409434188
It is a Mobile Number
5252552659
Invalid input.!!
02812452728
It is a Landline Number
df
Invalid input.!!
```

PRACTICAL-3

1. **Write a Lex program to recognize identifiers in C**
    - **C identifiers should have following constraint**
        - **It should start with either letter or underscore (_) sign.**
        - **It should not contain special symbols.**

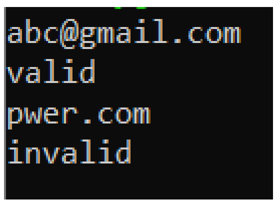
```
%{
#include<stdio.h>
%}
%%
^[a-zA-Z_][_a-zA-Z0-9]+ {printf("Identifier\n");}
.* {printf("Not a Identifier\n");}
%%
main()
{
yylex();
}
```



```
A1
Identifier

12
Not a Identifier

_12
Identifier

A_1
Identifier

A@a
Not a Identifier

var1
Identifier
```

2. Write a Lex program for validation of Email-Add. (Consider Email Add. from any domain e.g. @gmail.com)

```
%{
#include<stdio.h>
%}
%%
^[a-z][a-z0-9_]*(@[A-Za-z]+)(\.[a-z]+)+ {printf("valid");}
.* {printf("invalid");}
%%
main()
{
yylex();
}
```



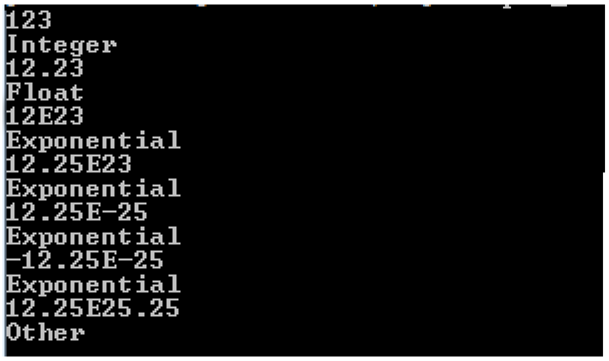
```
abc@gmail.com
valid
pwer.com
invalid
```

3. Write a Lex program to identify integer, float and exponential value.

Examples:

```
123
Integer
12.23
Float
12E23
Exponential
12.25E23
Exponential
12.25E-25
Exponential
-12.25E-25
Exponential
12.25E25.25
Other          (Reason: float value after E not allowed)
```

```
%{
#include<stdio.h>
%}
%%
[0-9]+printf("Integer");
[0-9]+\.[0-9]+ printf("Float");
[0-9]+E[?\-0-9][0-9]* printf("Exponential");
[?\-0-9][0-9]*?.[0-9]+E[?\-0-9][0-9]* printf("Exponential");
.*printf("Other");
%%
int yywrap(void)
{
return 1;
}
int main()
{
yylex();
return 0;
}
```



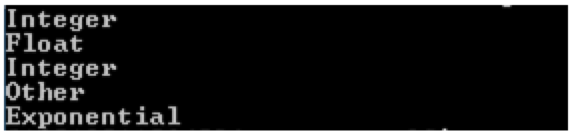
```
123
Integer
12.23
Float
12E23
Exponential
12.25E23
Exponential
12.25E-25
Exponential
-12.25E-25
Exponential
12.25E25.25
Other
```

4. **Implement a Lex program that takes input from user specified file and labels each input as Integer, Float, Exponential or Other.**

   Consider file test.txt contains following data:

   123
   123.25
   15
   Abs
   12.25E25

```
            Expected Output:
            123 - Integer
            123.25 - Float
            15 - Integer
            Abs - Other
            12.25E25 - Exponential
%{
#include<stdio.h>
%}
%%
  /*** Rules section ***/
[-0-9]* {
printf("Integer Value");
  }
^[-|0-9][0-9]+(.)[0-9]+ {
printf("Float Value");
  }
[-0-9]*(.)[0-9]+(E)[-0-9]+ {
printf("Exponential Value");
}
.* {
printf("Invalid value");
  }
%%
int main()
{
  FILE *fp;
fp = fopen("test1.txt", "r");
  if (fp == NULL) { printf("File not found"); }
yyin = fp;
yylex();
  return 0;
}
```

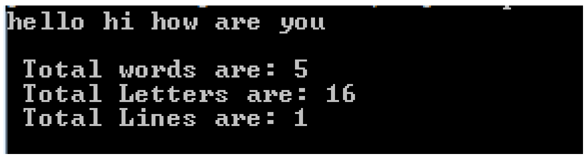


```
Integer
Float
Integer
Other
Exponential
```

PRACTICAL-4

1. **Write a lex program to count number of words, characters and lines from user input.**

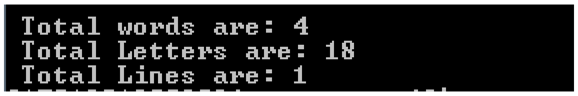
```
%{
#include<stdio.h>
int count=0;
int letter=0;
int lcount=0;
%}

%%
[a-zA-Z0-9] {letter++;}
" " {count++;}
"\n" {lcount++; count++; printf("\n Total words are: %d\n Total Letters are: %d\n Total
Lines are: %d\n",count,letter,lcount); lcount=0; count=0; letter=0;}
%%
int yywrap()
{
return 1;
}
int main()
{
yylex();
return 0;
}
```



```
hello hi how are you

 Total words are: 5
 Total Letters are: 16
 Total Lines are: 1
```

2. **Write a lex program to count number of words, characters and lines from File.**

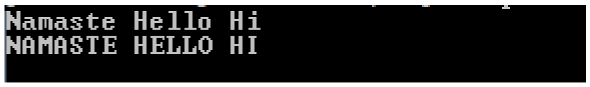
```
%{
#include<stdio.h>
int count=0;
int letter=0;
int lcount=0;
%}

%%
[a-zA-Z0-9] {letter++;}
" " {count++;}
"\n" {lcount++; count++; printf("\n Total words are: %d\n Total Letters are: %d\n Total
Lines are: %d\n",count,letter,lcount); lcount=0; count=0; letter=0;}
%%

int yywrap()
{
return 1;
}
int main()
{
yyin=fopen("test.txt","r");
yylex();
return 0;
}
```



```
Total words are: 4
Total Letters are: 18
Total Lines are: 1
```

3. **Write a lex program to convert lower case letter to upper case from user input and terminate the program if user enters 0.**

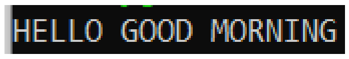
```
%{
#include<stdio.h>
%}

%%
[a-z] printf("%c",yytext[0] - ('a' - 'A'));
[A-Z] printf("%c",yytext[0] - ('A' - 'A'));
0 { return 0;}
%%

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



```
Namaste Hello Hi
NAMASTE HELLO HI
```

4. **Write a lex program to convert lower case letter of given file to upper case.**

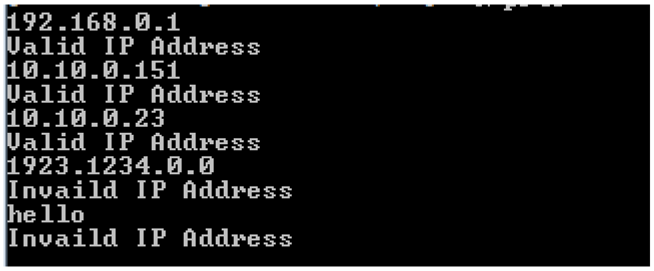
```
%{
#include<stdio.h>
%}

%%
 /*** Rules section ***/
[a-z] printf("%c",yytext[0] - ('a' - 'A'));
[A-Z] printf("%c",yytext[0] - ('A' - 'A'));
0 { return 0;}
%%

int main()
{
  FILE *fp;
  fp = fopen("test.txt", "r");
  if (fp == NULL) { printf("File not found"); }
  yyin = fp;

 yylex();
 return 0;
}
```



```
HELLO GOOD MORNING
```

5. Write a lex program to check whether IP address entered by user is valid or not.

```
%{
#include<stdio.h>
%}

%%
^(([0-9]|[1-9][0-9]|1[0-9]{2}|2[0-4][0-9]|25[0-5])\.){3}([0-9]|[1-9][0-9]|1[0-9]{2}|2[0-
4][0-9]|25[0-5])$ printf("Valid IP Address");
[a-zA-Z0-9.]* printf("Invaild IP Address");
0 { return 0;}
%%

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

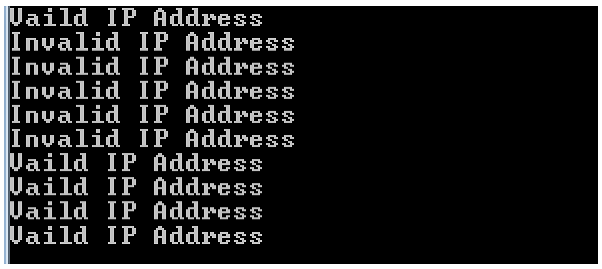


```
192.168.0.1
Valid IP Address
10.10.0.151
Valid IP Address
10.10.0.23
Valid IP Address
1923.1234.0.0
Invaild IP Address
hello
Invaild IP Address
```

6. **Write a lex program to validate IP addresses from user specified file.**

```
%{
#include<stdio.h>
%}

%%
```

```
^(([0-9]|[1-9][0-9]|1[0-9]{2}|2[0-4][0-9]|25[0-5])\.){3}([0-9]|[1-9][0-9]|1[0-9]{2}|2[0-
4][0-9]|25[0-5])$ printf("Vaild IP Address");
[A-Za-z0-9.]* printf("Invalid IP Address");
%%

int main()
{
  FILE *fp;
  fp = fopen("ipaddress.txt", "r");
  if (fp == NULL) { printf("File not found"); }
  yyin = fp;

 yylex();
 return 0;
}
```

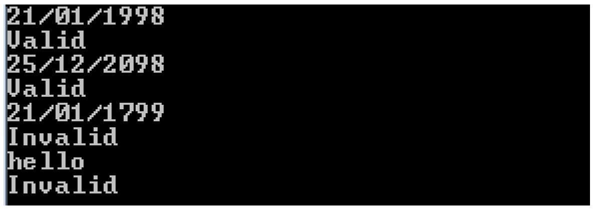


```
Vaild IP Address
Invalid IP Address
Invalid IP Address
Invalid IP Address
Invalid IP Address
Invalid IP Address
Vaild IP Address
Vaild IP Address
Vaild IP Address
Vaild IP Address
```

7. **Write a lex program to check whether Date entered by user in format (dd/mm/yyyy) is valid or not.**

```
%{
#include<stdio.h>
%}
```

```
%%
((0[1-9])|([1-2][0-9])|(3[0-1]))\/((0[1-9])|(1[0-2]))\/(19[0-9]{2}|2[0-9]{3})
printf("Valid");
.* printf("Invalid");
%%

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



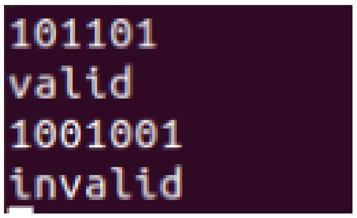
```
21/01/1998
Valid
25/12/2098
Valid
21/01/1799
Invalid
hello
Invalid
```

PRACTICAL-5

1. **The language of all strings contains exactly two 0's**

```
%{
#include<stdio.h>
%}
%%
1*01*01* printf("valid");
.* printf("invalid");
%%
int yywrap()
{
return 1;
}
int main()
{
yylex();
return 0;
}
```

OUTPUT



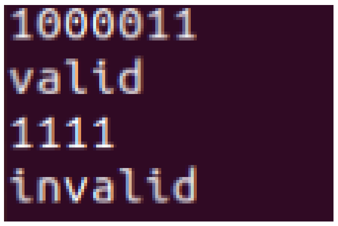
```
101101
valid
1001001
invalid
```

5. **The language of all strings contains atleast two 0's**

```
%{
#include<stdio.h>
%}
%%
(0|1)*0(0|1)*0(0|1)* printf("valid");
.* printf("invalid");
%%
int yywrap()
```

```
{
return 1;
}
int main()
{
yylex();
return 0;
}
```

OUTPUT

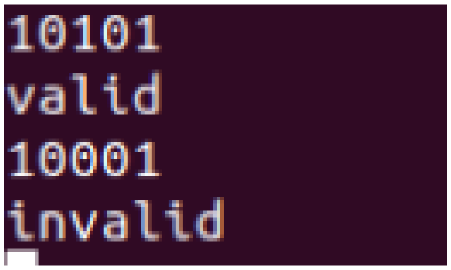


```
1000011
valid
1111
invalid
```

6. The language of all strings ending in 1 and not containing 00

```
%{
#include<stdio.h>
%}
%%
1|(1|01)*1* printf("valid");
.* printf("invalid");
%%
int yywrap()
{
return 1;
}
int main()
{
yylex();
return 0;
}
```

OUTPUT

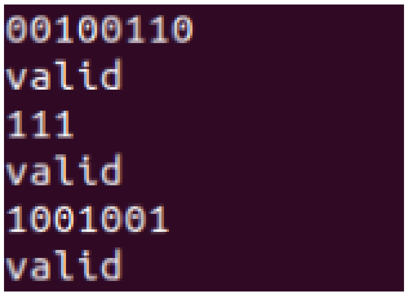


```
10101
valid
10001
invalid
```

7. String with odd number of 1's

```
%{
#include<stdio.h>
%}
%%
0*1(0|10*1|11)* printf("valid");
.* printf("invalid");
%%
int yywrap()
{
return 1;
}
int main()
{
yylex();
return 0;
}
```

OUTPUT



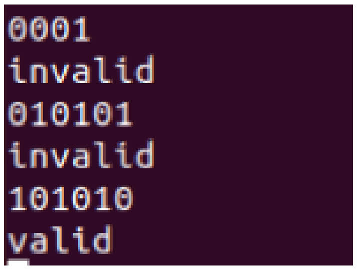
```
00100110
valid
111
valid
1001001
valid
```

8. The language of all strings that do not end with 01

```
%{
```

```
#include<stdio.h>
%}
%%
0|1|(0|1)*(00|10|11) printf("valid");
.* printf("invalid");
%%
int yywrap()
{
return 1;
}
int main()
{
yylex();
return 0;
}
```

OUTPUT



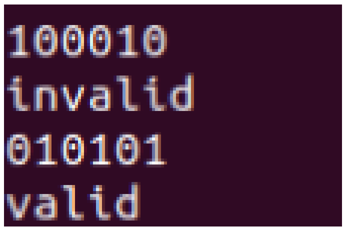
```
0001
invalid
010101
invalid
101010
valid
```

7. The language of all string not containing 00

```
%{
#include<stdio.h>
%}
%%
0|1|(1|010)* printf("valid");
.* printf("invalid");
%%
int yywrap()
{
return 1;
}
```

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

OUTPUT

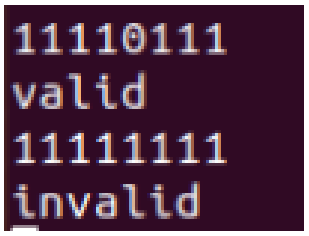


```
100010
invalid
010101
valid
```

7. The language of all string containing either 10 or 001

```
%{
#include<stdio.h>
%}
%%
(0|1)*(10|001)(0|1)* printf("valid");
.* printf("invalid");
%%
int yywrap()
{
return 1;
}
int main()
{
yylex();
return 0;
}
```

OUTPUT



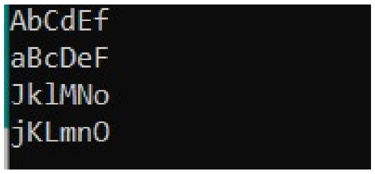
```
11110111
valid
11111111
invalid
```

PRACTICAL- 6

1. **Write a program to identify the word and change the case of each character of word (i.e if character in word is in lowercase then convert it to uppercase and if it is uppercase then convert to lowercase)**

```
%{
#include<stdio.h>
%}
%%
[a-z] {printf("%c",yytext[0]-32);}
[A-Z] {printf("%c",yytext[0]+32);}
%%
int yywrap()
{
  return 1;
}
int main()
{
        yylex();
}
```

OUTPUT



```
AbCdEf
aBcDeF
JklMNo
jKLmnO
```

2. **Write a Lex program to count number of comments (single line or multiple lines) in given C source file.**

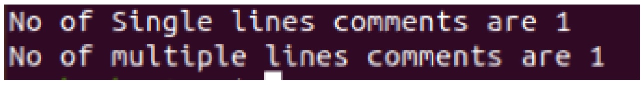
```
%{
     #include<stdio.h>
     int sl=0,ml=0;
%}
%%
"//" { sl++; }
"/*"([^*]|[*][^/])*"*/" { ml++; }
.|\n { }
%%
```

```
int yywrap()
{
	return 1;
}
int main()
{
	FILE *fp;
	fp=fopen("pr2.c","r");
	if(fp==NULL)
	{
		printf("Error");
		return 0;
	}
	yyin=fp;
	yylex();
	printf("No of Single lines comments are %d\n",sl);
	printf("No of multiple lines comments are %d\n",ml);

}
```

C File

```
#include<stdio.h>//  header file
#include<conio.h>
#define pi 3.14

void main()
{
	printf("Hello");
}
/*abc
{
}*/
```

OUTPUT

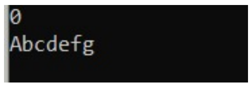


```
No of Single lines comments are 1
No of multiple lines comments are 1
```

3. Write a LEX program to display your name when 0 is entered and nothing should be displayed on the screen.

```
%{
        #include<stdio.h>
%}
%%
0  { printf("Dharmay"); }
.|\n  { }
%%
int yywrap()
{
  return 1;
}
int main()
{
        yylex();
}
```

OUTPUT



```
0
Abcdefg
```

4. Write a LEX program to count word "India" from file.

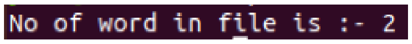
Expected Input from file: India is a great country and India have large population.

Expected output: word count = 2

```
%{
#include<stdio.h>
int sl=0;
%}
%%
India  { sl++; }
.|\n  ;
%%
int yywrap()
{
  return 1;
}
int main()
{
       FILE *fp;
      fp=fopen("india.txt","r");
      if(fp==NULL)
```

```
        {
                printf("Error");
                return 0;
        }
        yyin=fp;
        yylex();
        printf("No of word in file is :- %d\n",sl);
}
```

OUTPUT



```
No of word in file is :- 2
```

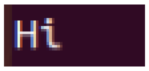
5. Display content of text file on output screen with a word "Hello" is replaced by "Hi".

```
%{
#include<stdio.h>
%}
%%
"Hello" {printf("Hi");}
%%
int yywrap()
{
  return 1;
}
int main()
{
FILE *f1;
        f1=fopen("hi.txt","r");
        if(f1==NULL)
        {
                printf("Error");
                return 0;
        }
        yyin=f1;
yylex();
}
```

File:

Hello

OUTPUT



```
Hi
```

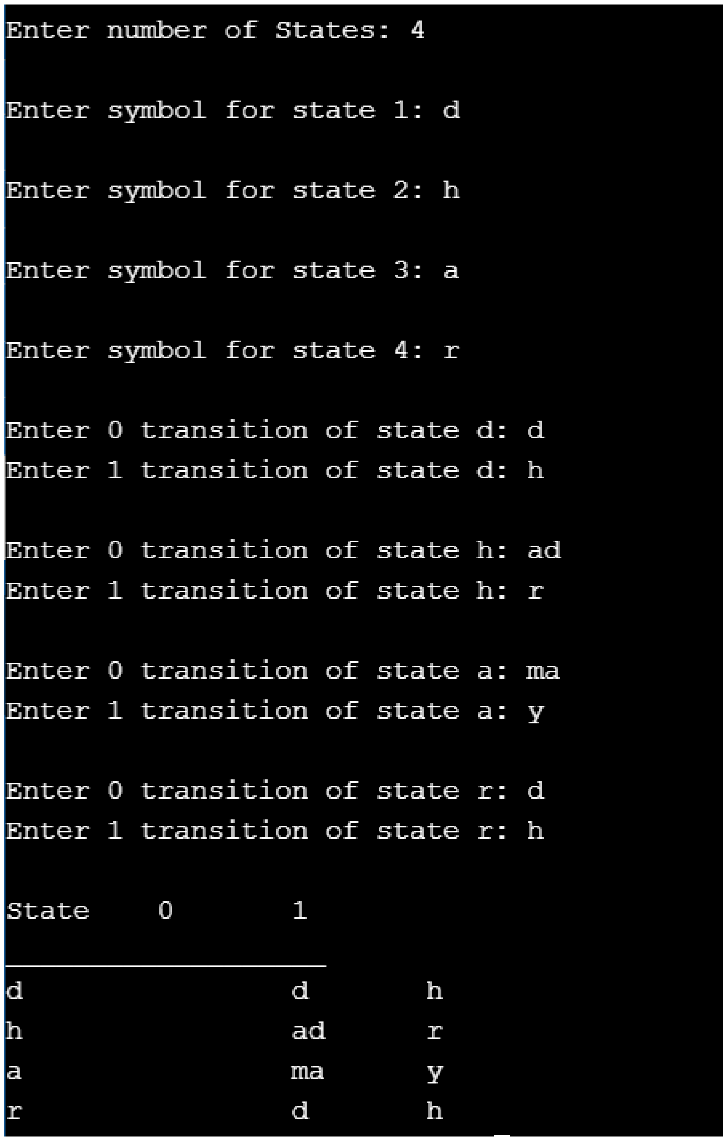
PRACTICAL-7

AIM : Given NFA for (11+110)*0 . Write a program to find δ* (q0, 110) and δ* (q0, 11)

Code:

```
#include<stdio.h>
#include<string.h>

int get_index(char states[],char a)
{
	int i;
	//check if state is present in set or not.. if present then return its index
	for(i=0;states[i]!='\0';i++)
	{
		if(states[i]==a)
		{
			return i;
		}
	}
	return -1;
}

void union_of_states(char src[],char des[])
{
	int i,len;

	for(i=0;src[i]!='\0';i++)
	{
		if(strchr(des,src[i])=='\0')
		{
			len=strlen(des);
			des[len]=src[i];
			des[len+1]='\0';
		}
	}
}


int main()
{
	int no_of_states,pos_of_state;
	char states[5];
char zero[5][5],one[5][5];
char temp1[5],temp2[5];
char str[10];
char initial_state,choice;

int i,j;

//get total no. of states in NFA
printf("Enter number of States: ");
scanf("%d",&no_of_states);


//get names of all states
for(i=0;i<no_of_states;i++)
{
        printf("\nEnter symbol for state %d: ",(i+1));
        scanf("\n%c",&states[i]);
}
states[i]='\0';


//get 0 and 1 transition of every state
for(i=0;i<no_of_states;i++)
{
                printf("\nEnter 0 transition of state %c: ",states[i]);
                scanf("%s",&zero[i]);

                printf("Enter 1 transition of state %c: ",states[i]);
                scanf("%s",&one[i]);
}

//printing the table
printf("\nState    0\t 1\n");
printf("____________________\n");
for(i=0;i<no_of_states;i++)
{
        printf("%c \t\t %s \t %s\n",states[i],zero[i],one[i]);
}

while(1)
{
        printf("Enter String to be checked.: ");
        scanf("%s",&str);
        temp1[0]=states[0];
        temp1[1]='\0';

        temp2[0]='\0';

        for(i=0;str[i]!='\0';i++)
        {
            temp2[0]='\0';
            for(j=0;temp1[j]!='\0';j++)
            {
                pos_of_state=get_index(states,temp1[j]);
                if(str[i]=='0')
                {
                    union_of_states(zero[pos_of_state],temp2);
                }
                else
                {
                    union_of_states(one[pos_of_state],temp2);
                }
            }
            strcpy(temp1,temp2);
        }
        printf("Final States are: %s\n",temp1);

        printf("\n\nCheck for new String?\nPress 'y' for yes and 'n' for no.\n");
        scanf("\n%c",&choice);
        if(choice=='n'||choice=='N')
        {
            return 0;
        }
    }
    return 0;
}
```

Output:



```
Enter number of States: 4

Enter symbol for state 1: d

Enter symbol for state 2: h

Enter symbol for state 3: a

Enter symbol for state 4: r

Enter 0 transition of state d: d
Enter 1 transition of state d: h

Enter 0 transition of state h: ad
Enter 1 transition of state h: r

Enter 0 transition of state a: ma
Enter 1 transition of state a: y

Enter 0 transition of state r: d
Enter 1 transition of state r: h

State    0       1

___________________
d                 d       h
h                 ad      r
a                 ma      y
r                 d       h
```