DOMS | Page No.
Date / /

eg. Write a regular definition for email id.

⇒ abc @ psit.ac.in.

digit → 0 | - - - - - | 9

letters → a | - - - | z | A | - - - | z

email → letters$^+$ '@' letters$^+$ '.' letters$^+$ '.' letters$^+$

Q- Write a regular definition for signed / unsigned number

eg: 5681 , 1234.552 , 12.34E+65 , 56.34E-65
123.45E20

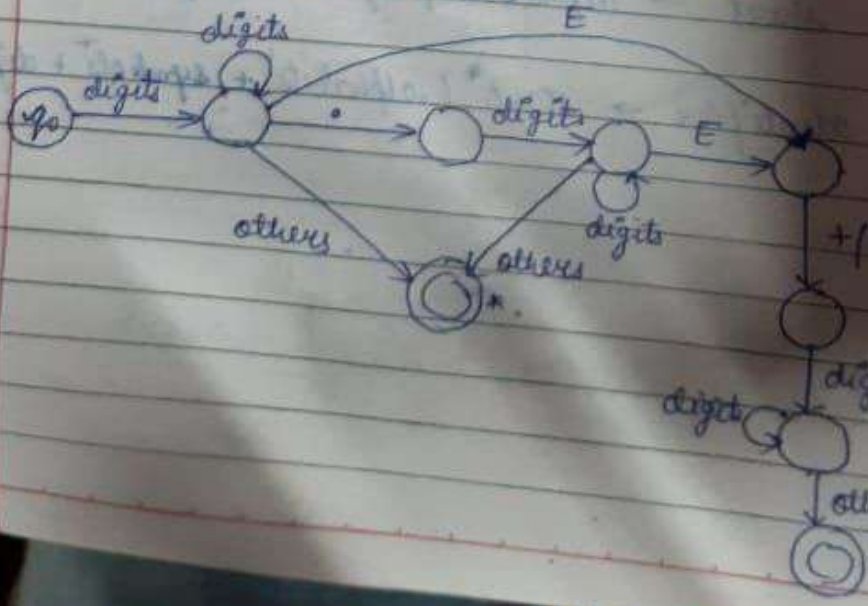
digits → 0 | - - - - - | 9

point → . digits → digits digits*

optional fraction → .digits | ε

optional exponent → (E (+|-|ε)) digits (ε)

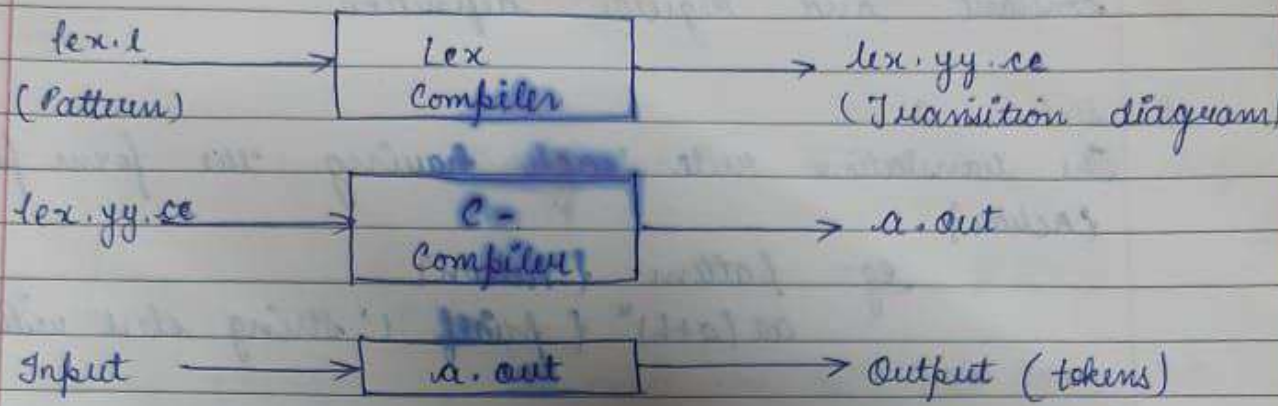
number → digits optional fraction optional exponent



6/Feb/18

DOM'S Page No. Date / /

Lex Compiler / Lex Language



⇒ Lex / Flex that allows a lexical analyser by specifying regular expression to describe pattern for tokens.

⇒ The input notation for the lex tool (lex compiler) is known as lex language.

⇒ Lex Compiler transform the input pattern into a transition diagram and generate code in a file lex.yy.c that contains transition diagram.

⇒ Structure of lex program is -

Section 1 [... Declaration ...] / Definition

Section 2 [... Transition Rules ...]

Section 3 [... auxiliary function ...] / Subroutine

Section 1
The declaration section includes declaration of variable, constant and regular definition.

Section 2
The translation rule each having the form pattern {Action}.

eg: pattern {Action}
aa(a+b)* { printf ("string start with aa"); }

Each pattern is a regular expression which may use regular definition of the declaration section.

The action are written in a C language form.

Section 3
It holds whatever additional functions are used in the action.

These function can be compiled seperately and load with lexical analyser.

Program-1

```
%{
    #include <stdio.h>          // Header File
%}
    // Next write regular definition (if any)
// Transition Rule
%%
    "HELLO" { printf ("GoodBye"); }
%%
```

// Lex function start with yylex for token production and lexical compiler get started.

```
main()
  {
    yylex();        // scan input and store in its buffer.
  }
```

Program-2

String having regular expression as aab (a+b+c)*

Regular definition > letter → a|b|c or letter → [a-c]

Solution

```
%{
    #include <stdio.h>
%}
    letter → a|b|c

%%
    aab {letter}* { printf ("string starts with aab"); }
%%

main()
  {
    yylex();
  }
```

7/Feb/18

Program 3

Q. Write a lex program to count no of vowels and consonants in a given string.

Solution:

```
%{
    #include <stdio.h>
    int c=0, v=0
%}

%%
    [aeiouAEIOU] { v++; }
    [A-Za-z]     { c++; }
%%
main()
{
    printf("Enter any string");
    yylex();
    printf("No. of vowels = %d", v);
    printf("No. of consonants = %d", c);
}
```

Program 4

Q. Write a lex program to accept length of string as 2 and containing only a and b.

Regular expression ⇒ (a+b)(a+b)

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

```
%%
{a|b}{a|b} { printf ("string has length of 2");}
%%

main ()
 {
   yylex ();
 }
```

Program-5

Q- Write a lex program to accept string of length 2 with a, b used as a the string must contain a substring as aab.

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

%%
{a/b} {a/b} { printf ("string length is exactly 2");}
{a|b}* {a} {a} {b} {a|b}* { printf (" The string has substring
 aab ");}
%%

main ()
 {
   yylex ();
 }
```