3/8/2020
Thursday

Lex Syntax and Example:-

Lex takes an input file containing a set of lexical analysis rules or regular expressions. For output, Lex produces a C function which when invoked, finds the next match in the input stream.

(i) Format of Lex input

```
declarations
%%
token-rules (or) Translation rules
%%
Auxiliary procedures.
```

(ii) Declarations:-

(a) String Sets;
(b) Standard C;

```
name Character-class
%{ -- c declarations --
%}
```

(iii) Token rules:-

```
regular-expression { optional C-code }
```

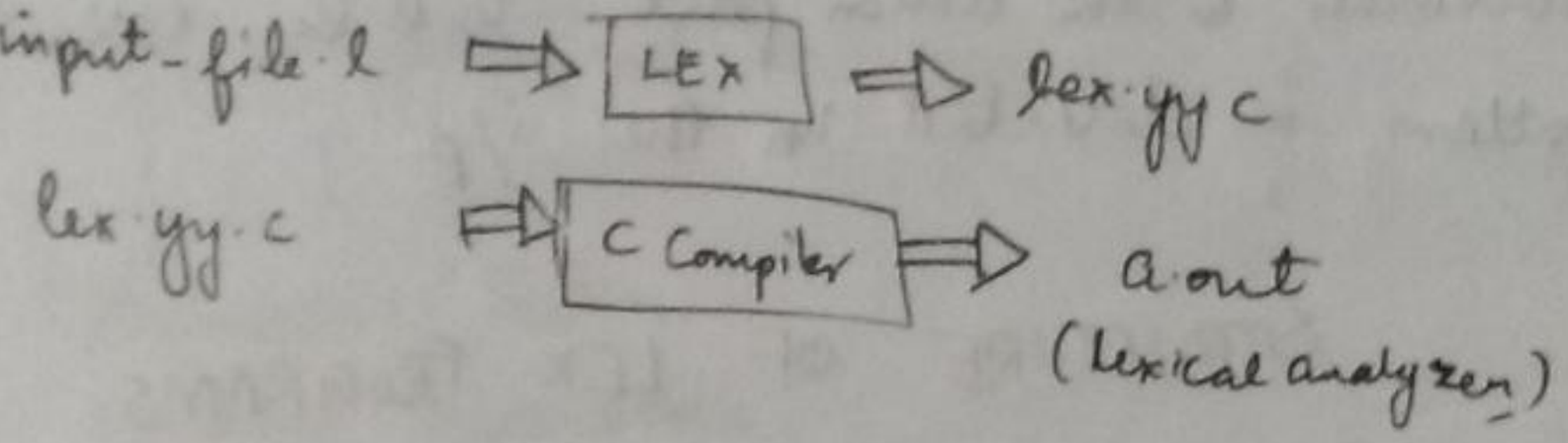
(a) If the regular expression includes a reference to the character class, include the class name in brackets { }

Regular Expression operators:-

* `*, +` → closure, positive closure
* `" ", \` → protection of special chars
* `|` → or
* `^` → Beginning of line anchor
* `()` → Grouping
* `$` → End of line anchor.
* `?` → Zero or one
* `.` → Any char (except \n)
* `{ref}` → Reference to a named character class
* `[ ]` → character class
* `[^ ]` → Not Character class

USING LEX

Lex is a tool used to generate a lexical analyzer. LEX translates a set of regular expression specifications given as I/p in input_file.l into a C implementation of a corresponding Finite State Machine (lex.yy.c). This C program when compiled, yields an executable lexical analyzer.



The source ExpL pgm is fed as i/p to lexical analyzer which produces a sequence of tokens as o/p. A lexical analyzer scans a given source ExpL pgm and produces an o/p of tokens.

Each token is specified by a token name. Its an abstract symbol representing the kind of lexical unit. The token names are i/p symbols that the parser processes. Eg: Integer, boolean, begin, end, if, which are tokens in ExpL.

i) Procedure to compile and execute LEX program:-

Vi filename.l

lex filename.l

cc lex.yy.c -ll

./a.out

Vi filename.l

lex filename.l

~~cc lex.yy.c -~~

CC -O filename lex.yy.c -ll

(or) ~~CC lex.yy.c -O~~

(or) CC lex.yy.c -ll -o filename -ll

YACC

For running Yacc program alone:-

```
vi filename.y
yacc filename.y
cc y.tab.c -ly
./a.out
```

1. Check if a given identifier is valid or not.

```
%{
#include<stdio.h>
#include<string.h>
%}
letter [A-Za-z_]
digit [0-9]
id ^{letter}({letter}|{digit})*
```

srihari@LAPTOP-AJMTFS87: ~/compiler_design/week1

```
srihari@LAPTOP-AJMTFS87:~/compiler_design/week1$ ./a.out
^C
srihari@LAPTOP-AJMTFS87:~/compiler_design/week1$ ./a.out
identifiercheck
Valid Identifier:identifiercheck

123h
Invalid Identifier
Srihari_85
Valid Identifier:Srihari_85

:klm
Invalid Identifier
^C
srihari@LAPTOP-AJMTFS87:~/compiler_design/week1$ cat validId.l
%{
#include<stdio.h>
#include<string.h>
%}
letter [A-Za-z_]
digit [0-9]
id ^{letter}({letter}|{digit})*

%%
{id} {printf("Valid Identifier:%s\n",yytext);}
^[^A-Za-z_] {printf("Invalid Identifier");}
. ;
%%


int main(){
        yylex();
        return 0;
}
srihari@LAPTOP-AJMTFS87:~/compiler_design/week1$
```

```
%%
{id}  {printf("valid Identifier : %s\n", yytext);}
^[^A-Za-z_] {printf("Invalid Identifier");}
.;
%%

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

OUTPUT

```
123h
Invalid Identifier
Sribani_85
Valid Identifier: Sribani_85
```

2. To show the use of yylex write a program, if it finds a number.

```
%option noyywrap
%{
#include<stdio.h>
#include<stdlib.h>
%}
number [0-9]+

%%
{number} {return atoi(yytext);}
%%

int main(){
    int num = yylex();
    printf("Found %d", num);
    return 1;
}
```

OUTPUT

```
89
Found 89
a
a
34
Found 34
```

3. Count the number of digits in a number.

```
%option noyywrap
%{
#include<stdio.h>
#include<stdlib.h>
%}
number [0-9]+
```

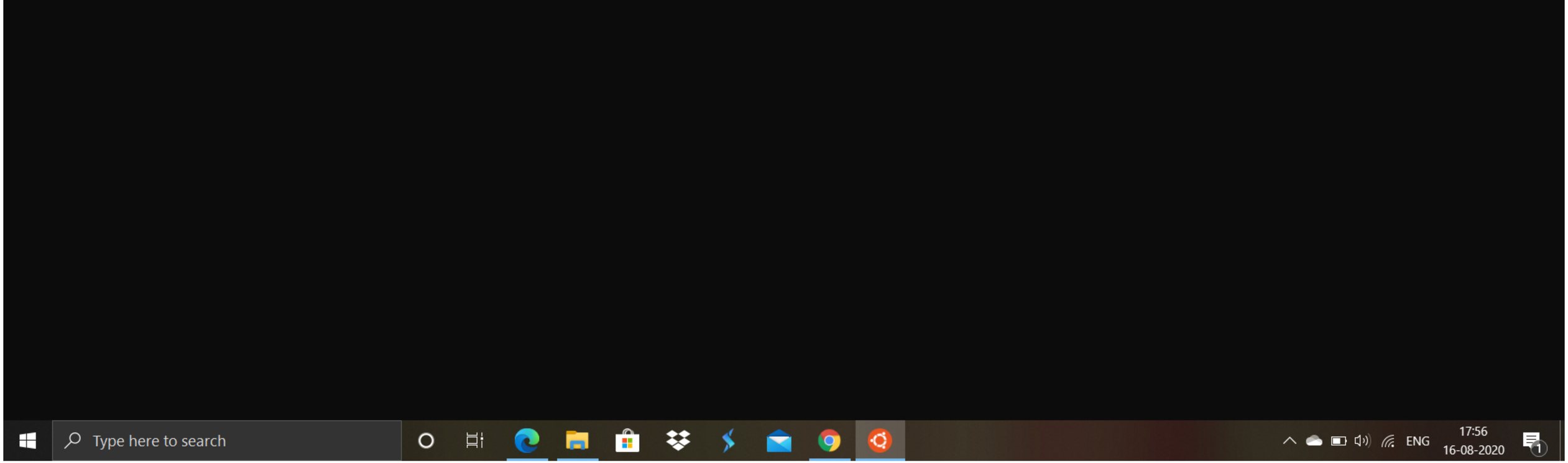
OUTPUT

```
98
Number of digits = 2
03451
Number of digits = 5
```

srihari@LAPTOP-AJMTFS87: ~/compiler_design/week1

```
srihari@LAPTOP-AJMTFS87:~/compiler_design/week1$ cat pgm1.l
%option noyywrap
%{
#include<stdio.h>
#include<stdlib.h>
%}
number [0-9]+

%%
{number} {return atoi(yytext);}
%%
int main(){
        int num = yylex();
        printf("Found %d",num);
        return 1;
}
srihari@LAPTOP-AJMTFS87:~/compiler_design/week1$ ./a.out
Srihari
Srihari
89
Found 89srihari@LAPTOP-AJMTFS87:~/compiler_design/week1$ ./a.out
34
Found 34srihari@LAPTOP-AJMTFS87:~/compiler_design/week1$
```



srihari@LAPTOP-AJMTFS87: ~/compiler_design/week1

```
srihari@LAPTOP-AJMTFS87:~/compiler_design/week1$ ./a.out
9876543212
Number of digits =10
98
Number of digits =2
ab
ab
0
Number of digits =1
^C
srihari@LAPTOP-AJMTFS87:~/compiler_design/week1$ cat pgm2.l
%option noyywrap
%{
#include<stdio.h>
#include<stdlib.h>
%}
number [0-9]+
%%
{number} {printf("Number of digits =%d",yyleng);}
%%

int main(){
        yylex();
        return 0;
}
srihari@LAPTOP-AJMTFS87:~/compiler_design/week1$
```

```
%.%.
{number}  {printf("Number of digits = %d", yyleng);}
%.%

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

5. Count the #words, Characters and Sentences in a paragraph.

```
%{
#include <stdio.h>
#include <string.h>
int i=0, c=0, s=0;
%}
%%
([a-zA-Z0-9])* {i++; c+=yyleng;}
[.!?] {s++;}
"\n" {if (s==0) s++; printf("word Count = %d \t Character Count = %d
      \t Sentence Count = %d \n", i, c, s); i=0; c=0; s=0;}
%%
```

srihari@LAPTOP-AJMTFS87: ~/compiler_design/week1

```
srihari@LAPTOP-AJMTFS87:~/compiler_design/week1$ ls
a.l  a.out  lex.yy.c  numop.l  pgm1.l  pgm2.l  pgm3.l  validId.l  wordCount.l
srihari@LAPTOP-AJMTFS87:~/compiler_design/week1$ lex wordCount.l
srihari@LAPTOP-AJMTFS87:~/compiler_design/week1$ cc lex.yy.c -ll
srihari@LAPTOP-AJMTFS87:~/compiler_design/week1$ ./a.out
I'm trying to count the number of words. After that I could count chars! Finally you would see sentence count as 3.
'                     word count = 23    character count = 90    sentence count = 3
^C
srihari@LAPTOP-AJMTFS87:~/compiler_design/week1$ cat wordCount.l
%{
#include<stdio.h>
#include<string.h>
int i=0,c=0,s=0,s1=0;
%}

%%
([a-zA-Z0-9])* {i++;c += yyleng; }
[.!?] {s++;}
"\n" {if(s == 0)s++;printf("word count = %d \t character count = %d \t sentence count = %d \n",i,c,s); i=0;c=0;s=0;}
%%

int yywrap(void){}
int main(){
        yylex();
        return 0;
}

srihari@LAPTOP-AJMTFS87:~/compiler_design/week1$
```

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

OUTPUT

```
Counting words!
WordCount = 2   CharacterCount = 14
                SentenceCount = 1
```

6. Finding numbers and operators

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

number [0-9]+
op [-|+|*|/|^|=]

%%
[a-zA-Z]+ {printf("Word");}
{number}  {printf(" number");}
{op}      {printf(" operator");}
%%

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

OUTPUT

```
23
number
4 + 3
number operator number
```

7. Program which terminates on encountering specific characters.

```
D [0-9]
A [a-zA-Z]
%{
#include<stdio.h>
#define NUM 1
#define ID 2
#define PLUS 3
#define ASSIGN 5
#define MINUS 4
#define SEMI 6
%}
```

srihari@LAPTOP-AJMTFS87: ~/compiler_design/week1

```
srihari@LAPTOP-AJMTFS87:~/compiler_design/week1$ ls
a.l  a.out  lex.yy.c  numop.l  validId.l  wordCount.l
srihari@LAPTOP-AJMTFS87:~/compiler_design/week1$ vim a.l
srihari@LAPTOP-AJMTFS87:~/compiler_design/week1$ lex a.l
srihari@LAPTOP-AJMTFS87:~/compiler_design/week1$ cc lex.yy.c -ll
srihari@LAPTOP-AJMTFS87:~/compiler_design/week1$ ./a.out
23
 number
4+7
 numberoperator number
^C
srihari@LAPTOP-AJMTFS87:~/compiler_design/week1$ cat a.l
%{
#include <stdio.h>
int global_variable;
%}

number [0-9]+
op [-|+|*|/|^|=]
%%
{number} {printf(" number");}
{op} {printf("operator");}
%%

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

srihari@LAPTOP-AJMTFS87:~/compiler_design/week1$
```

```
%{
%}
{D}+ return NUM;
{A}({A}|{D})* return ID;
"+" return PLUS;
"-" return MANUS;
";" return SEMI;
":=" return ASGN;
. ;
%%

int main(){
    yylex();
    return 0;
}
void yyerror(){
    printf("error");
    exit(0);
}
```

OUTPUT

```
+
-
check
```

8. Find an integer.

```
%{
#include<stdio.h>
%}
%option noyywrap
%%
[0-9]+ {printf("Saw Integer: %s \n", yytext);}
.|\n {}
%%
int main(void){
        yylex();
        return 0;
}
```

OUTPUT

```
23
Saw Integer: 23
we
71h
Saw Integer: 71
```

srihari@LAPTOP-AJMTFS87: ~/compiler_design/week1

```
srihari@LAPTOP-AJMTFS87:~/compiler_design/week1$ cat pgm6.l
D [0-9]
A [a-zA-Z]
%{
#include<stdio.h>
#define NUM 1
#define ID 2
#define PLUS 3
#define MINUS 4
#define ASGN 5
#define SEMI 6
%}

%%
{D}+ return NUM;
{A}({A}|{D})* return ID;
"+" return PLUS;
"-" return MINUS;
";" return SEMI;
:= return ASGN;
. ;
%%

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

void yyerror(){
        printf("error");
        exit(0);
}
srihari@LAPTOP-AJMTFS87:~/compiler_design/week1$ ./a.out
*


_

check
srihari@LAPTOP-AJMTFS87:~/compiler_design/week1$
```

srihari@LAPTOP-AJMTFS87: ~/god

```
srihari@LAPTOP-AJMTFS87:~$ ls
compiler_design  god
srihari@LAPTOP-AJMTFS87:~$ cd god
srihari@LAPTOP-AJMTFS87:~/god$ ls
a.c  a.out  first.l  lex.yy.c
srihari@LAPTOP-AJMTFS87:~/god$ cat first.l
%{
#include<stdio.h>
%}
%option noyywrap
%%
[0-9]+ {
        printf("Saw integer:%s\n",yytext);}
.|\n { }
%%
int main(void){
        yylex();
return 0;
}
srihari@LAPTOP-AJMTFS87:~/god$ lex first.l
srihari@LAPTOP-AJMTFS87:~/god$ cc lex.yy.c -ll
srihari@LAPTOP-AJMTFS87:~/god$ ./a.out
23
Saw integer:23
a1
Saw integer:1
we
a
71h7
Saw integer:71
Saw integer:7
^C
srihari@LAPTOP-AJMTFS87:~/god$
```