OUTPUT:

```
lex count.l
gcc lex.yy.c -o count -ll
./count filename.txt
```

For an input file text.txt with content like

```
Hello World
This is a test.
```

OUTPUT:

```
Lines : 2
Blank Spaces : 6
Words : 5
```

POORNIMA

Experiment – 6

Objective: Write a LEX program to count blank spaces, words, lines in a given file.

About the program: The lex program is an example of how you can use regular expressions to count specific features (such as blank spaces) in a text file. The program efficiently processes input files and outputs the desired counts.

Procedure:

1) Install Lex: First, make sure you have Lex installed on your system.
2) Create the lex file: Write the lex program in .l file
3) Generate C code: Use lex to generate C code from the lex file.
4) Compile the C program: Use a C compiler to compile the generated C code.

Program:

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

POORNIMA

```
int blank_count = 0;
int word_count = 0;
int line_count = 0;
%}
%%
\n      { line_count++; }
[[:space:]]+ { blank_count++; }
[a-zA-Z]+ { word_count++; }
%%
int main(int argc, char *argv[]){
if (argc != 2){
printf("Usage: %s <filename>\n", argv[0]);
return 1;
}
FILE *file = fopen(argv[1], "r");
if (file == NULL){
perror("Error opening file");
return 1;
}
yyin = file
yylex();
printf("Lines: %d\n" line_count);
printf("Blank Spaces: %d\n", blank_count);
printf("Words: %d\n", word_count);
fclose(file);
return 0;
}
```

Sample OUTPUT

C file (example.c) with content

```
#include <stdio.h>
int main() {
  printf("Hello, world !\n");
  return 0;
}
```

⇒ OUTPUT :

Vowels : 7

Consonants : 10

POORNIMA

Experiment - 07

Objective : Write a Lex program to count no. of vowels and consonants in a C file.

About the program : The Lex program counts the number of vowels and consonants in a C file. It outputs the counts of vowels and consonants in the file.

Procedure :

1) Install Lex :
2) Write the Lex Program
3) Generate C code from lex
4) Compile the C Program.
5) Run the Program : Execute the compiled Program with a C file as input to count the vowels and consonants.

Program : (count_vowels_consonant.l)

```
%{
#include <stdio.h>
#include <ctype.h>
int vowel_count = 0;
int consonant_count = 0;
%}
```

Page No.....

POORNIMA

```
%.%
[aAeEiIoOuU]    {Vowel_count++;}
[b-df-hj-np-tv-zB-DF-HJ-NP-TV-Z]
  {
    Consonant_count++;}
%.%
int main(int args, char *argv[]) {
if (argc != 2) {
printf("Usage : %.s <file name>\n",
      argv[0]);
return 1;
}
file *file = fopen [argv[I], "r");
 if (file == NULL) {
perror ("Error opening file");
return 1;
}
yyin = file
 yylex();
printf("Vowels : %d\n", Vowel_count);
printf(" consonants : %.d\n", Consonant_count);
  fclose(file);
    return 0;
  }
```

POORNIMA

Experiment - 8

Objective : Write a LEX program to identify following

1) Valid Mobile Number
2) Valid URL
3) Valid Identifiers
4) Valid date (dd/mm/yyyy)
5) Valid time (hh:mm:ss)

About the Program:

1) Valid mobile : A 10-digit number starting with digits 7, 8 or 9
2) Valid URL : Starts with http:// or https:// followed by alphanumeric characters, optional periods, slashes and extensions.
3) Valid identifier : Start with a letter or underscore and followed by letters, digits or underscore.

Procedure:

1) Start
2) Define rules for:
   - Mobile numbers starting with 7, 8 or 9
   - URLs beginning with https:// or http.

Input :

9876543210

https : // example . com

- valid identifiers 123

15 / 08 / 2024

12 : 45 : 03

Expected Output

Valid Mobile Number : 9876 543210

Valid URL : https : // example . com

Valid Identifier : valid identifier 123

Valid Date : 15 / 08 / 2024

Valid Time : 12 : 45 : 03

- identifiers that start with a letter or underscore.
- dates in dd / mm / yy format
- time in hh : mm : ss format

Program :

```
%{
#include <stdio.h>
%}
%%
[7-9][0-9]{9}  { printf("Valid Mobile Number : %s \n", yytext); }

(https|https) : // [a-zA-Z0-9.]+
    { printf("Valid URL : %s \n",
    yytext); }
[_a-zA-Z][_a-zA-Z0-9]*
  { printf("Valid Identifier : %s \n",
  yytext); }

([0][1-9]|[12][0-9]|3[01])/([0][1-9]|
    1[0-2])/[0-9]{4}
{ printf("Valid Date : %s \n", yytext); }

([01][0-9]|2[0-3]):[0-5][0-9]:
  [0-5][0-9]
  { printf("Valid Time : %s \n", yytext); }
%%
```

Page No.....

POORNIMA

```
int main () {
printf (" Enter text to validate : \n") ;
  yylex ();
return 0;
    }
```

28/11/24

POORNIMA

Experiment - 09

Objective : Write a program to find first from a given grammar.

About the program : The first set of a grammar helps to determine which terminal symbols can appear at the beginning of strings derived from the non-terminal.

Algorithm:

1) Input the grammar
- Read the grammar rules, which consist of non terminals and productions.

2) Initialize first Set :
- Create empty FIRST sets for each non terminal.

Program:

```
#include<stdio.h>
#include<string.h>
#include<ctype.h>
#define MAX 10
char productions[MAX][MAX], first[MAX][MAX];
int n;
```

POORNIMA

```
void findfirst(char symbol, char result[]);
void addToResultSet(char result[], char value[]);
int main(){
 int i;
char result[MAX];
printf("Enter the number of productions:");
scanf("%d", &n);
printf("Enter the productions (E->AB/a, use /
for multiple productions):\n");
for(i=0; i<n; i++){
 scanf("%s", production[i]);
}
for(i=0; i<n; i++){
char non_Terminal = production[i][0];
find first(non-Terminal, result);
strcpy(first[i], result);
}
printf("\n FIRST ids =\n");
for(i=0; i<n; i++){
printf("FIRST (%c) = {%s}\n", production[i]
        [0], first[i]);
   }
return 0;
}
void findfirst(char symbol, char result[]){
 int i, j, foundEpsilon;
char subResult[MAX];
result[0] = '\0';
```

Page No.

Input :→
Number of productions : 3
Productions :
E → TR
T → FS
F → a

Output →
FIRST (E) = {a}
FIRST (T) = {a}
FIRST (F) = {a}

POORNIMA

```
if (!isupper(Symbol)){
addToResultSet(result, symbol);
return ;}
for (i=0; i<n; i++){
if (productions[i][0] == Symbol){
char current = production[i][j];
FoundEpsilon = 0
for (j=3; production[i][j] != '\0'; j++){
char current = production[i][j];
if (current == '|')
continue;
}
findfirst(current, subresult);
Strcat(result, subresult);
if (strchr(subResult, 'e')
FoundEpsilon = 1;
} else{
FoundEpsilon = 0;
break;
}}
if(FoundEpsilon){
addToResultSet(result, 'e');
}}}}
void addToResult(char result[], char value){
if (strchr(result, value) == NULL){
int len = strlen(result);
result[len] = value;
result[len+1] = '\0';
}}
```