Experiment 3

Aim:

Implement Lexical Analyzer using FLEX

a. Count no of Vowels & Consonants
b. Count no of Words, characters & lines
c. Count no of keywords,identifiers & operators
d. Identify Even & odd integers
e. Count of printf & scanf statements in C program
f. Classify English words as verbs, adverbs, adjectives etc
g. Append line no to every line of code
h. Replace printf by write and scanf by read
i. Remove comments from source code

Theory:

What is LEX ?

LEX is a tool used for generating lexical analyzers (scanners) in compiler design. It takes a set of regular expressions as input and produces a C program that can recognize and tokenize input text based on those expressions. It is commonly used to process structured text, such as programming languages, configuration files, and structured data formats.

Explain a simple LEX Program with an example.

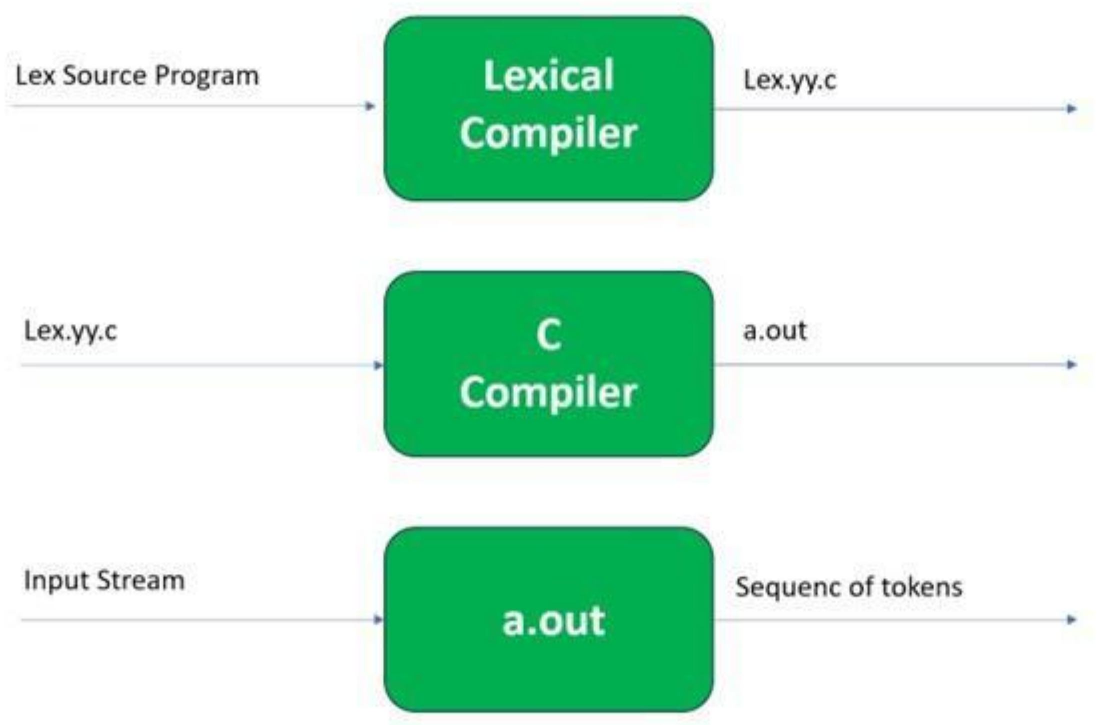
```
%{
#include <stdio.h>
%}
%%
[0-9]+     { printf("NUMBER: %s\n", yytext); }
[a-zA-Z]+  { printf("WORD: %s\n", yytext); }
.          { printf("SPECIAL CHARACTER: %s\n", yytext); }
%%
int main() {
   yylex();
   return 0;
}
int yywrap() { return 1; }
```

Explanation:

- **Definitions (%{ %} section)** – Includes C header files.

- **Rules (%% section)** – Defines patterns and corresponding actions (e.g., recognizing numbers, words, and special characters).
- **User code (%% section below rules)** – Contains the main() function that calls yylex() to start tokenizing input.

Block Diagram to explain the working of LEX.



Regular Expressions in LEX.

LEX uses regular expressions to define patterns for token recognition:

- [0-9]+ → Matches a number
- [a-zA-Z]+ → Matches a word
- \+, -, *, / → Matches operators
- "if", "else" → Matches keywords

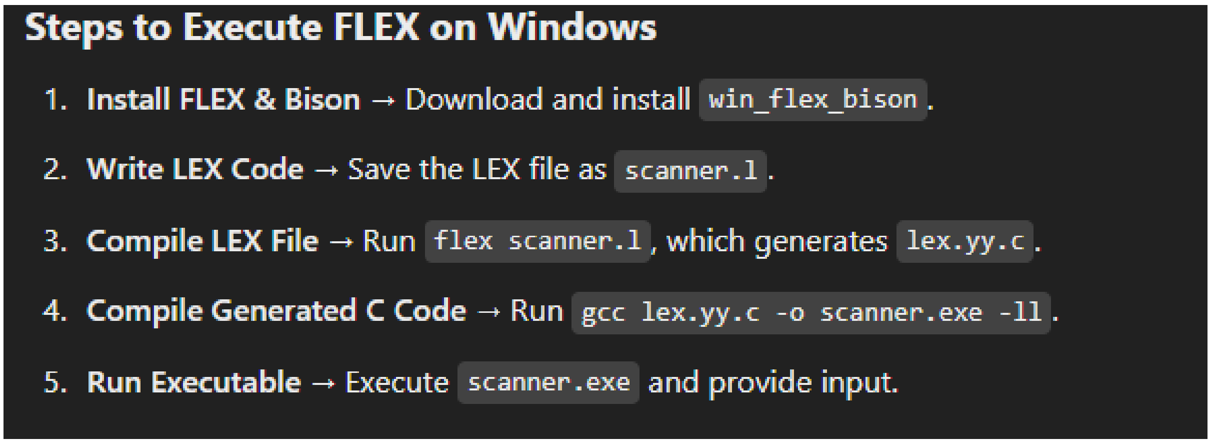
Pattern Matching Primitives.

- ^ → Matches the beginning of a line
- $ → Matches the end of a line
- \b → Matches a word boundary
- \s → Matches whitespace

LEX Predefined variables & Library routines.

- yytext → Stores the matched token
- yyleng → Stores the length of yytext
- yyin → Input file pointer
- yyout → Output file pointer
- yylex() → Main function for scanning tokens
- yywrap() → Called when input is exhausted

Steps for executing FLEX on windows



Steps to Execute FLEX on Windows

1. **Install FLEX & Bison** → Download and install `win_flex_bison`.
2. **Write LEX Code** → Save the LEX file as `scanner.l`.
3. **Compile LEX File** → Run `flex scanner.l`, which generates `lex.yy.c`.
4. **Compile Generated C Code** → Run `gcc lex.yy.c -o scanner.exe -ll`.
5. **Run Executable** → Execute `scanner.exe` and provide input.

1) Count no of Vowels & Consonants

Input.txt

There are many variations of passages of Lorem Ipsum available, but the majority have suffered alteration in some form, by injected humour, or randomised words which don't look even slightly believable. If you are going to use a passage of Lorem Ipsum, you need to be sure there isn't anything embarrassing hidden in the middle of text. All the Lorem Ipsum generators on the Internet tend to repeat predefined chunks as necessary, making this the first true generator on the Internet. It uses a dictionary of over 200 Latin words, combined with a handful of model sentence structures, to generate Lorem Ipsum which looks reasonable. The generated Lorem Ipsum is therefore always free from repetition, injected humour, or non-characteristic words etc.

Code:

```
%{
int vowels = 0;
int consonants = 0;
%}

%%
[aeiouAEIOU] { vowels++; }
[a-zA-Z] { consonants++; }
[^a-zA-Z] ;
%%

int yywrap(){ return 1; }

int main(int argc, char *argv[]) {
  FILE *fp;

  if (argc < 2) {
    printf("Usage: %s <filename>\n", argv[0]);
    return 1;
  }
  fp = fopen(argv[1], "r");
  if (!fp) {
    printf("Error Occurred: Unable to open file\n");
    return 1;
  }

  yyin = fp;
  yylex(); // Process the file
  fclose(fp);

  printf("Processing Input File: %s\n", argv[1]);
  printf("No of Vowels = %d\n", vowels);
  printf("No of Consonants = %d\n", consonants);

  return 0;
}
```

Output:

2) Count no of Words, characters and lines.

Input.txt

There are many variations of passages of Lorem Ipsum available, but the majority have suffered alteration in some form, by injected humour, or randomised words which don't look even slightly believable.

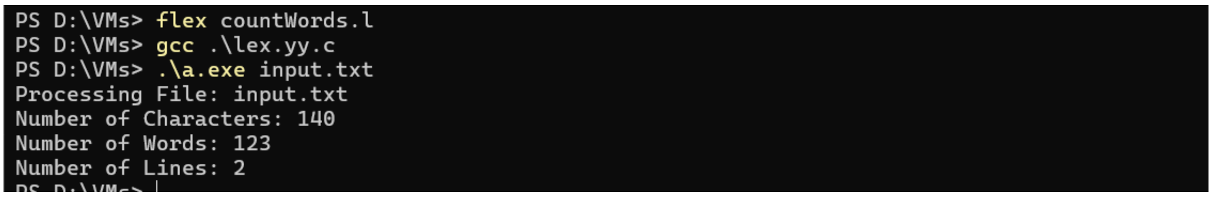
If you are going to use a passage of Lorem Ipsum, you need to be sure there isn't anything embarrassing hidden in the middle of text. All the Lorem Ipsum generators on the Internet tend to repeat predefined chunks as necessary, making this the first true generator on the Internet.

It uses a dictionary of over 200 Latin words, combined with a handful of model sentence structures, to generate Lorem Ipsum which looks reasonable. The generated Lorem Ipsum is therefore always free from repetition, injected humour, or non-characteristic words etc.

Code:

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

int words = 0, characters = 0, lines = 0;
extern FILE *yyin;
%}

%%
[a-zA-Z]+      { words++; }
.      { characters++; }
\n     { lines++; }

%%

int yywrap() { return 1; }

int main(int argc, char *argv[]) {
  FILE *fp;

  if (argc < 2) {
    printf("Usage: %s <filename>\n", argv[0]);
    return 1;
  }

  fp = fopen(argv[1], "r");
  if (!fp) {
    printf("Error: Unable to open file %s\n", argv[1]);
    return 1;
  }

  yyin = fp;
  yylex(); // Process file
  fclose(fp);

  printf("Processing File: %s\n", argv[1]);
  printf("Number of Characters: %d\n", characters);
  printf("Number of Words: %d\n", words);
  printf("Number of Lines: %d\n", lines);

  return 0;
}
```

Output:



```
PS D:\VMs> flex countWords.l
PS D:\VMs> gcc .\lex.yy.c
PS D:\VMs> .\a.exe input.txt
Processing File: input.txt
Number of Characters: 140
Number of Words: 123
Number of Lines: 2
PS D:\VMs>
```

3) Count no of keywords, identifiers and operators.

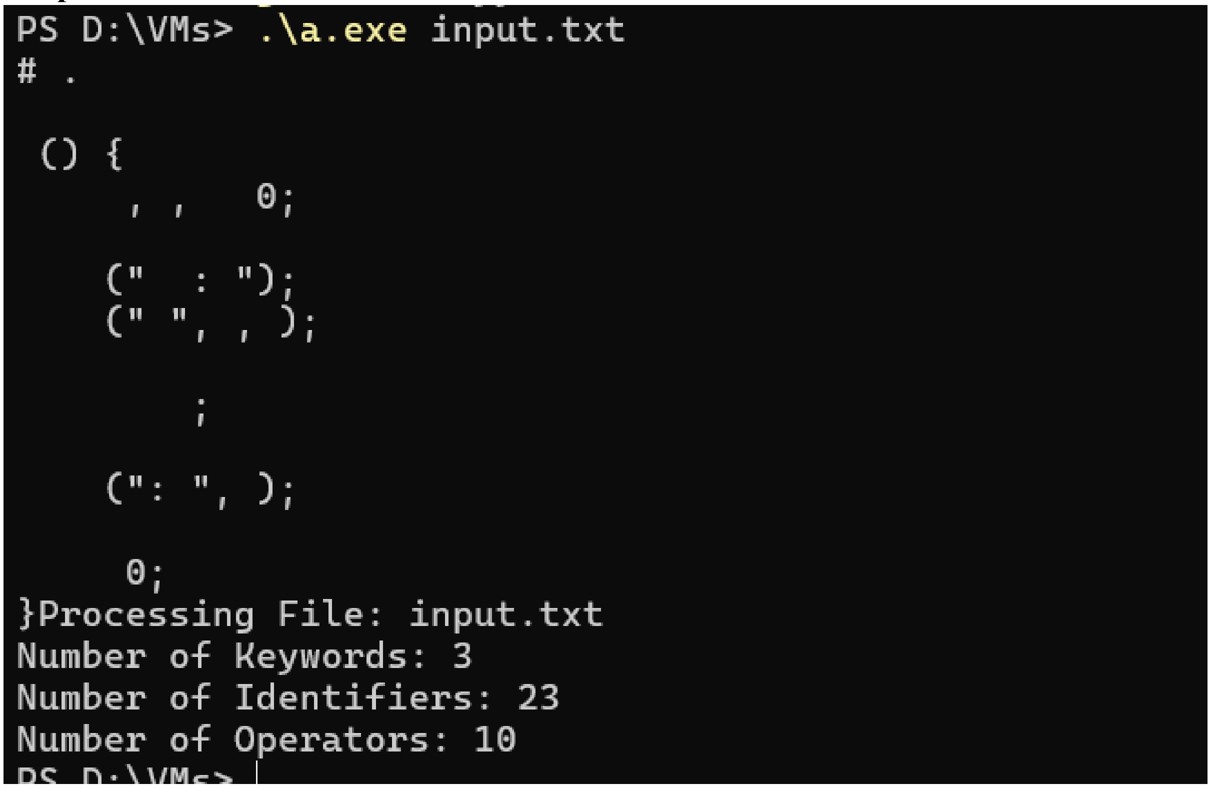
Input.txt:

```
#include <stdio.h>

int main() {
    int a, b, sum = 0;

    printf("Enter two integers: ");
    scanf("%d %d", &a, &b);

    sum = a + b;

    printf("Sum: %d", sum);

    return 0;
}
```

Code:

```
%{
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

int keywords = 0, identifiers = 0, operators = 0;
extern FILE *yyin;

// List of C keywords
char *keyword_list[] = {
    "auto", "break", "case", "char", "const", "continue",
"default", "do",
    "double", "else", "enum", "extern", "float", "for",
"goto", "if", "inline",
    "int", "long", "register", "restrict", "return", "short",
"signed",
    "sizeof", "static", "struct", "switch", "typedef",
"union", "unsigned",
    "void", "volatile", "while", "_Alignas", "_Alignof",
"_Atomic",
    "_Bool", "_Complex", "_Generic", "_Imaginary",
"_Noreturn",
    "_Static_assert", "_Thread_local"
};

int is_keyword(char *word) {
    int i;
    for (i = 0; i < sizeof(keyword_list) /
sizeof(keyword_list[0]); i++) {
        if (strcmp(keyword_list[i], word) == 0) {
            return 1;
        }
    }
    return 0;
}
%}

%%
[a-zA-Z_][a-zA-Z0-9_]* {
    if (is_keyword(yytext)) {
        keywords++;
    } else {
        identifiers++;
    }
}
[+\-*/=<>!&|%^] { operators++; }

%%

int yywrap() { return 1; }

int main(int argc, char *argv[]) {
  FILE *fp;

  if (argc < 2) {
    printf("Usage: %s <filename>\n", argv[0]);
    return 1;
  }

  fp = fopen(argv[1], "r");
  if (!fp) {
    printf("Error: Unable to open file %s\n", argv[1]);
    return 1;
  }

  yyin = fp;
  yylex(); // Process file
  fclose(fp);

  printf("Processing File: %s\n", argv[1]);
  printf("Number of Keywords: %d\n", keywords);
  printf("Number of Identifiers: %d\n", identifiers);
  printf("Number of Operators: %d\n", operators);

  return 0;
}
```

Output:

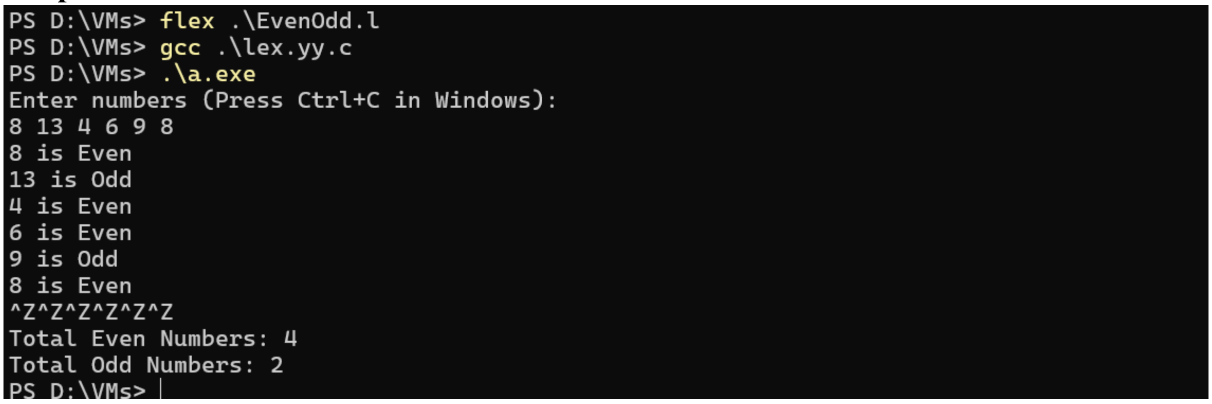


```
PS D:\VMs> .\a.exe input.txt
# .

 () {
     , ,   0;

    ("  : ");
    (" ", , );

        ;

    (": ", );

     0;
}Processing File: input.txt
Number of Keywords: 3
Number of Identifiers: 23
Number of Operators: 10
PS D:\VMs>
```

4) Identify Even and odd integers
Code:

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

int even_count = 0, odd_count = 0;
%}

%%
[0-9]*[02468] {
   printf("%s is Even\n", yytext);
   even_count++;
}

[0-9]*[13579] {
   printf("%s is Odd\n", yytext);
   odd_count++;
}

[ \t\n] ;

%%

int yywrap() { return 1; }

int main() {
   printf("Enter numbers (Press Ctrl+C in Windows):\n");

   yylex(); // Process user input

   printf("\nTotal Even Numbers: %d\n", even_count);
   printf("Total Odd Numbers: %d\n", odd_count);

   return 0;
}
```

Output:



```
PS D:\VMs> flex .\EvenOdd.l
PS D:\VMs> gcc .\lex.yy.c
PS D:\VMs> .\a.exe
Enter numbers (Press Ctrl+C in Windows):
8 13 4 6 9 8
8 is Even
13 is Odd
4 is Even
6 is Even
9 is Odd
8 is Even
^Z^Z^Z^Z^Z^Z
Total Even Numbers: 4
Total Odd Numbers: 2
PS D:\VMs> |
```

5) Count no of printf and scanf statements in C program

Input.txt

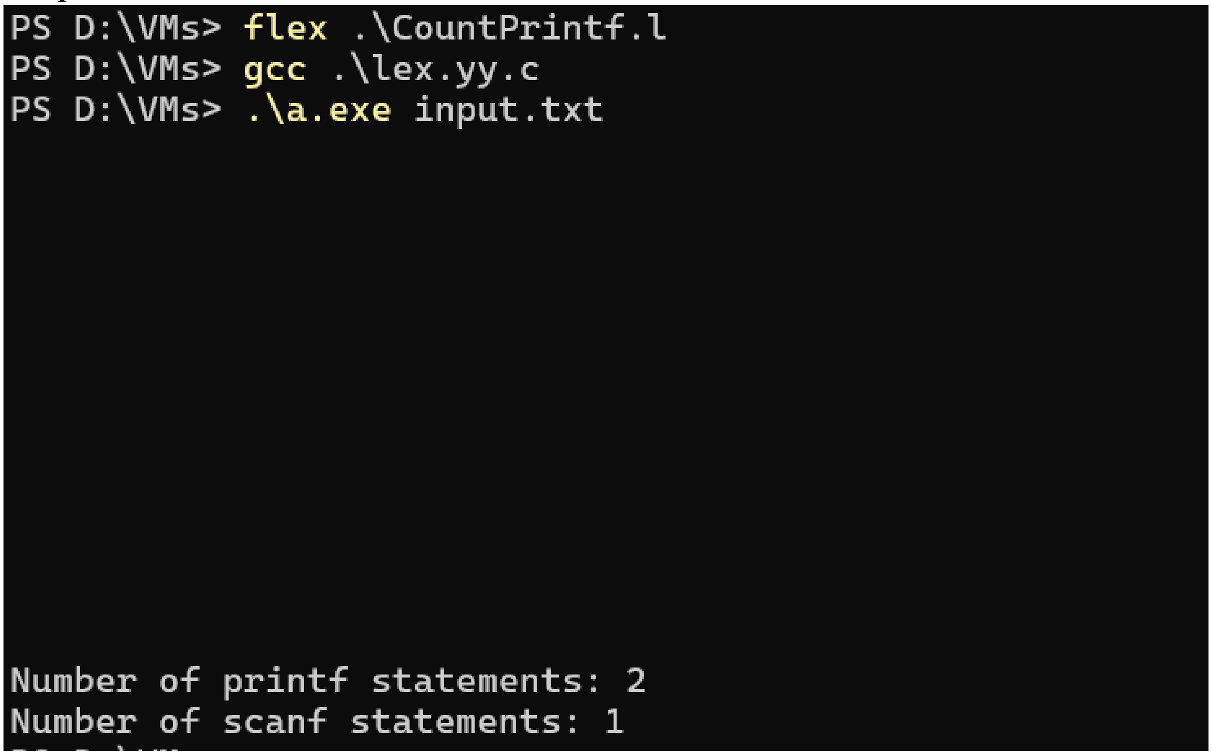
```
#include <stdio.h>

int main() {
   int a, b, sum = 0;

   printf("Enter two integers: ");
   scanf("%d %d", &a, &b);

   sum = a + b;

   printf("Sum: %d", sum);

   return 0;
}
```

Code

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

int printf_count = 0, scanf_count = 0;
%}

%%
printf { printf_count++; }

scanf { scanf_count++; }

. ;

%%

int yywrap() { return 1; }

int main(int argc, char *argv[]) {

   FILE *fp;

   if (argc < 2) {
      printf("Usage: %s <C program file>\n", argv[0]);
      return 1;
   }

   fp = fopen(argv[1], "r");
   if (!fp) {
      printf("Error: Unable to open file %s\n", argv[1]);
      return 1;
   }

   yyin = fp;
   yylex(); // Process the file
   fclose(fp);
```

```
    printf("Number of printf statements: %d\n", printf_count);
    printf("Number of scanf statements: %d\n", scanf_count);
    return 0;
}
```

Output



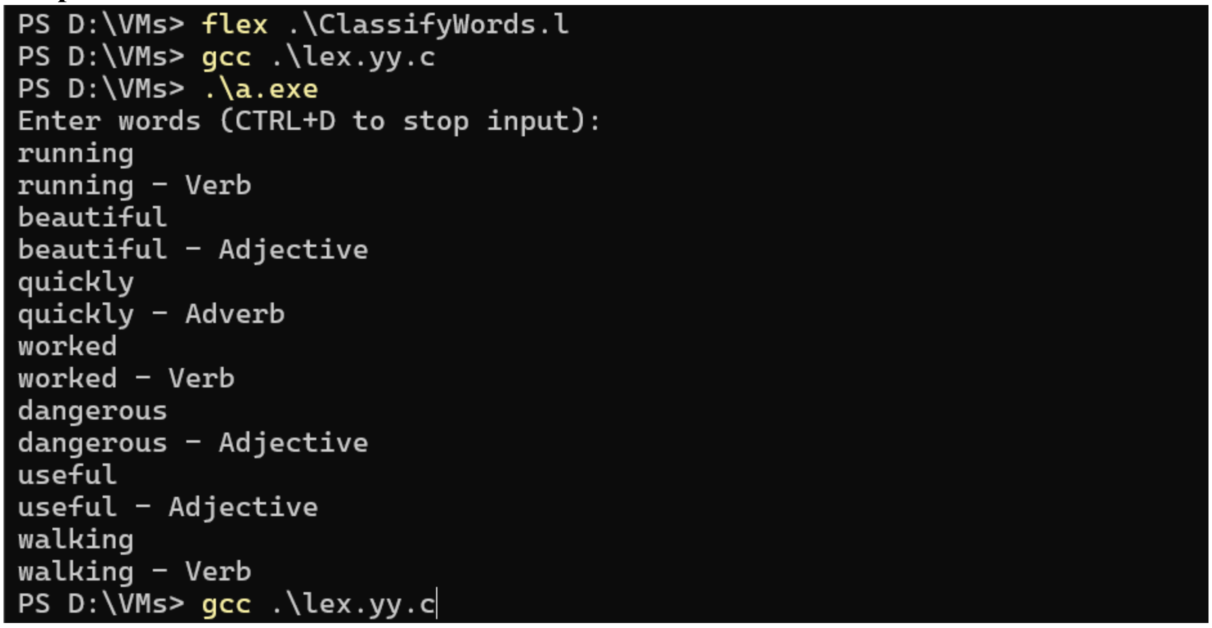
```
PS D:\VMs> flex .\CountPrintf.l
PS D:\VMs> gcc .\lex.yy.c
PS D:\VMs> .\a.exe input.txt








Number of printf statements: 2
Number of scanf statements: 1
```

6) Classify English Words as verbs, adverbs, adjectives.

Code:

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

int verbs = 0, adverbs = 0, adjectives = 0;
%}

%%

(.*ing|.*ed|.*es) { verbs++; printf("%s - Verb\n", yytext); }

(.*ly) { adverbs++; printf("%s - Adverb\n", yytext); }

(.*ous|.*ful|.*able|.*ive|.*ic) { adjectives++; printf("%s - Adjective\n", yytext); }

.|\n ;

%%

int yywrap() { return 1; }

int main() {
    printf("Enter words (CTRL+D to stop input):\n");
    yylex();

    printf("\nSummary:\n");
    printf("Verbs: %d\n", verbs);
    printf("Adverbs: %d\n", adverbs);
    printf("Adjectives: %d\n", adjectives);

    return 0;
}
```

Output:



```
PS D:\VMs> flex .\ClassifyWords.l
PS D:\VMs> gcc .\lex.yy.c
PS D:\VMs> .\a.exe
Enter words (CTRL+D to stop input):
running
running - Verb
beautiful
beautiful - Adjective
quickly
quickly - Adverb
worked
worked - Verb
dangerous
dangerous - Adjective
useful
useful - Adjective
walking
walking - Verb
PS D:\VMs> gcc .\lex.yy.c
```

7) Append line no to every line of code

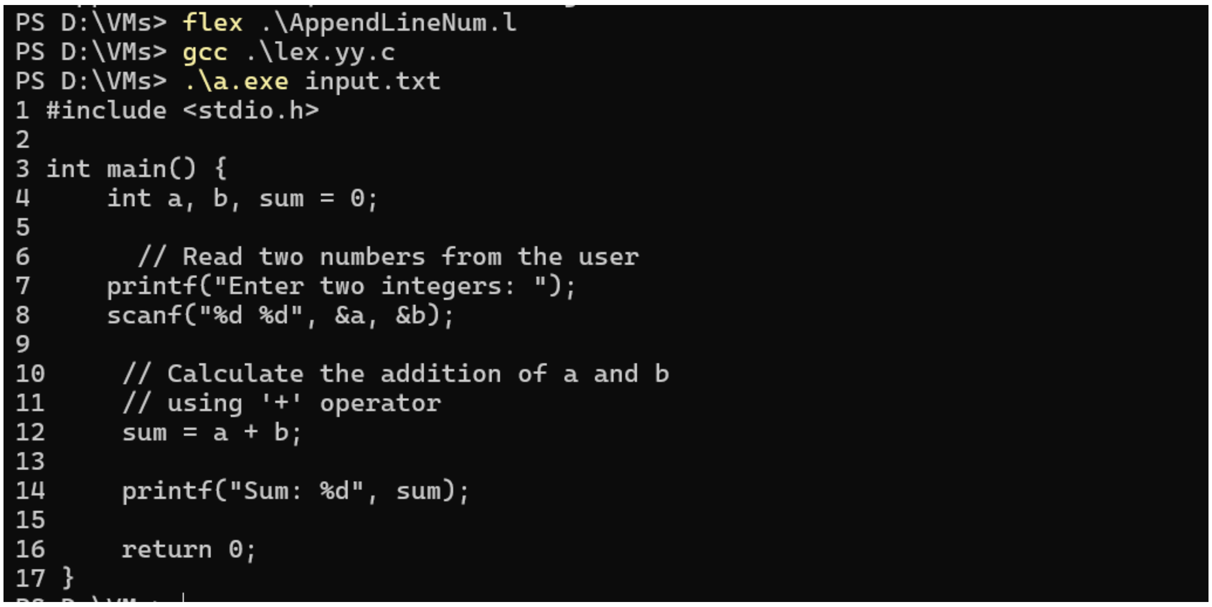
Input.txt:

```
#include <stdio.h>

int main() {
   int a, b, sum = 0;

     // Read two numbers from the user
   printf("Enter two integers: ");
   scanf("%d %d", &a, &b);

   // Calculate the addition of a and b
   // using '+' operator
   sum = a + b;

   printf("Sum: %d", sum);

   return 0;
}
```

Code:

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

int line_no = 1;
%}

%%

\n { printf("\n%d ", ++line_no); }

. { printf("%s", yytext); }

%%

int yywrap() { return 1; }

int main(int argc, char *argv[]) {
   FILE *fp;

   if (argc < 2) {
      printf("Usage: %s <filename>\n", argv[0]);
      return 1;
   }

   fp = fopen(argv[1], "r");
   if (!fp) {
      printf("Error: Unable to open file %s\n",
argv[1]);
      return 1;
   }

   yyin = fp;
   printf("1 ");   // Print the first line number
before processing
   yylex();
   fclose(fp);

   return 0;
}
```

Output:



```
PS D:\VMs> flex .\AppendLineNum.l
PS D:\VMs> gcc .\lex.yy.c
PS D:\VMs> .\a.exe input.txt
1 #include <stdio.h>
2
3 int main() {
4     int a, b, sum = 0;
5
6       // Read two numbers from the user
7     printf("Enter two integers: ");
8     scanf("%d %d", &a, &b);
9
10     // Calculate the addition of a and b
11     // using '+' operator
12     sum = a + b;
13
14     printf("Sum: %d", sum);
15
16     return 0;
17 }
```

8) Replace printf by write and scanf by read

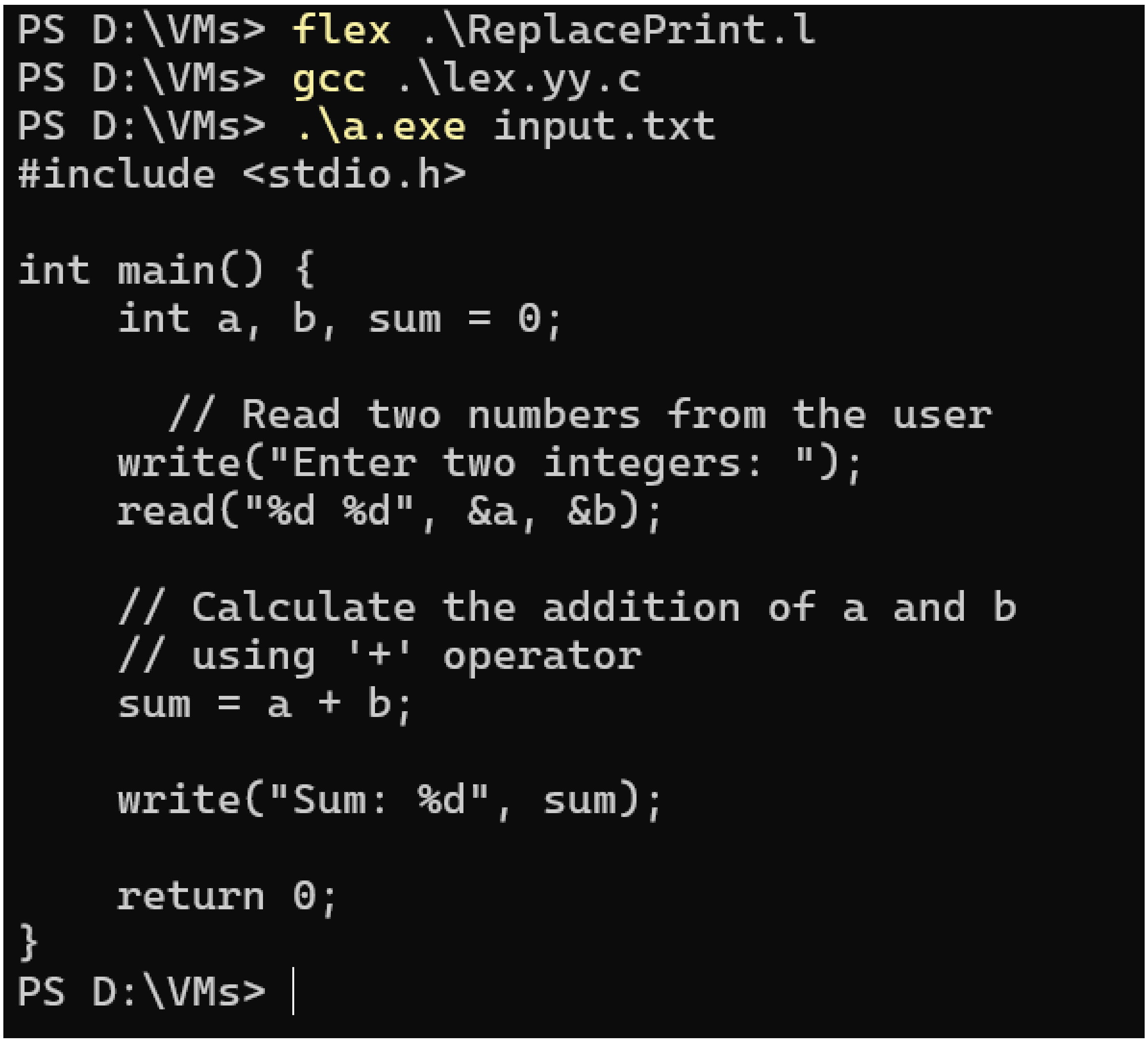
Input.txt

```
#include <stdio.h>

int main() {
    int a, b, sum = 0;

      // Read two numbers from the user
    printf("Enter two integers: ");
    scanf("%d %d", &a, &b);

    // Calculate the addition of a and b
    // using '+' operator
    sum = a + b;

    printf("Sum: %d", sum);

    return 0;
}
```

Code:

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

%}

%%
printf { printf("write"); }

scanf { printf("read"); }

. { printf("%s", yytext); }

%%

int yywrap() { return 1; }

int main(int argc, char *argv[]) {
    FILE *fp;

    if (argc < 2) {
        printf("Usage: %s <filename>\n", argv[0]);
        return 1;
    }

    fp = fopen(argv[1], "r");
    if (!fp) {
        printf("Error: Unable to open file %s\n",
argv[1]);

        return 1;
    }

    yyin = fp;
    yylex();
    fclose(fp);

    return 0;
}
```

Output:



```
PS D:\VMs> flex .\ReplacePrint.l
PS D:\VMs> gcc .\lex.yy.c
PS D:\VMs> .\a.exe input.txt
#include <stdio.h>

int main() {
    int a, b, sum = 0;

      // Read two numbers from the user
    write("Enter two integers: ");
    read("%d %d", &a, &b);

    // Calculate the addition of a and b
    // using '+' operator
    sum = a + b;

    write("Sum: %d", sum);

    return 0;
}
PS D:\VMs>
```

9) Remove comments from source code

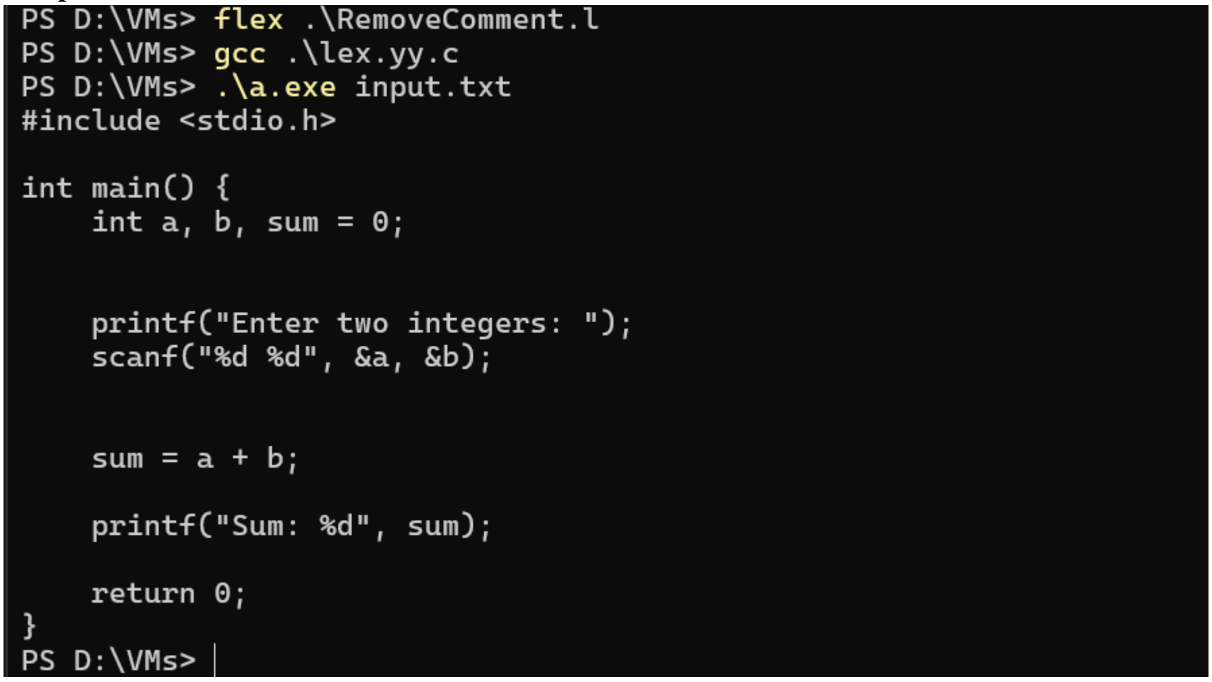
Input.txt:

```
#include <stdio.h>

int main() {
    int a, b, sum = 0;

     // Read two numbers from the user
    printf("Enter two integers: ");
    scanf("%d %d", &a, &b);

    /* Calculate the addition of a and b
     using '+' operator*/
    sum = a + b;

    printf("Sum: %d", sum);

    return 0;
}
```

Code:

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

%}

%%

\/\/.*  { /* Ignore */ }

\/\*[^*]*\*+([^/*][^*]*\*+)*\/  { /* Ignore */ }

. { printf("%s", yytext); }

%%

int yywrap() { return 1; }

int main(int argc, char *argv[]) {
    FILE *fp;

    if (argc < 2) {
        printf("Usage: %s <filename>\n", argv[0]);
        return 1;
    }

    fp = fopen(argv[1], "r");
    if (!fp) {
        printf("Error: Unable to open file %s\n",
argv[1]);
        return 1;
    }

    yyin = fp;
    yylex();
    fclose(fp);

    return 0;
}
```

Output:



```
PS D:\VMs> flex .\RemoveComment.l
PS D:\VMs> gcc .\lex.yy.c
PS D:\VMs> .\a.exe input.txt
#include <stdio.h>

int main() {
    int a, b, sum = 0;


    printf("Enter two integers: ");
    scanf("%d %d", &a, &b);


    sum = a + b;

    printf("Sum: %d", sum);

    return 0;
}
PS D:\VMs>
```

10) Count No of Comments

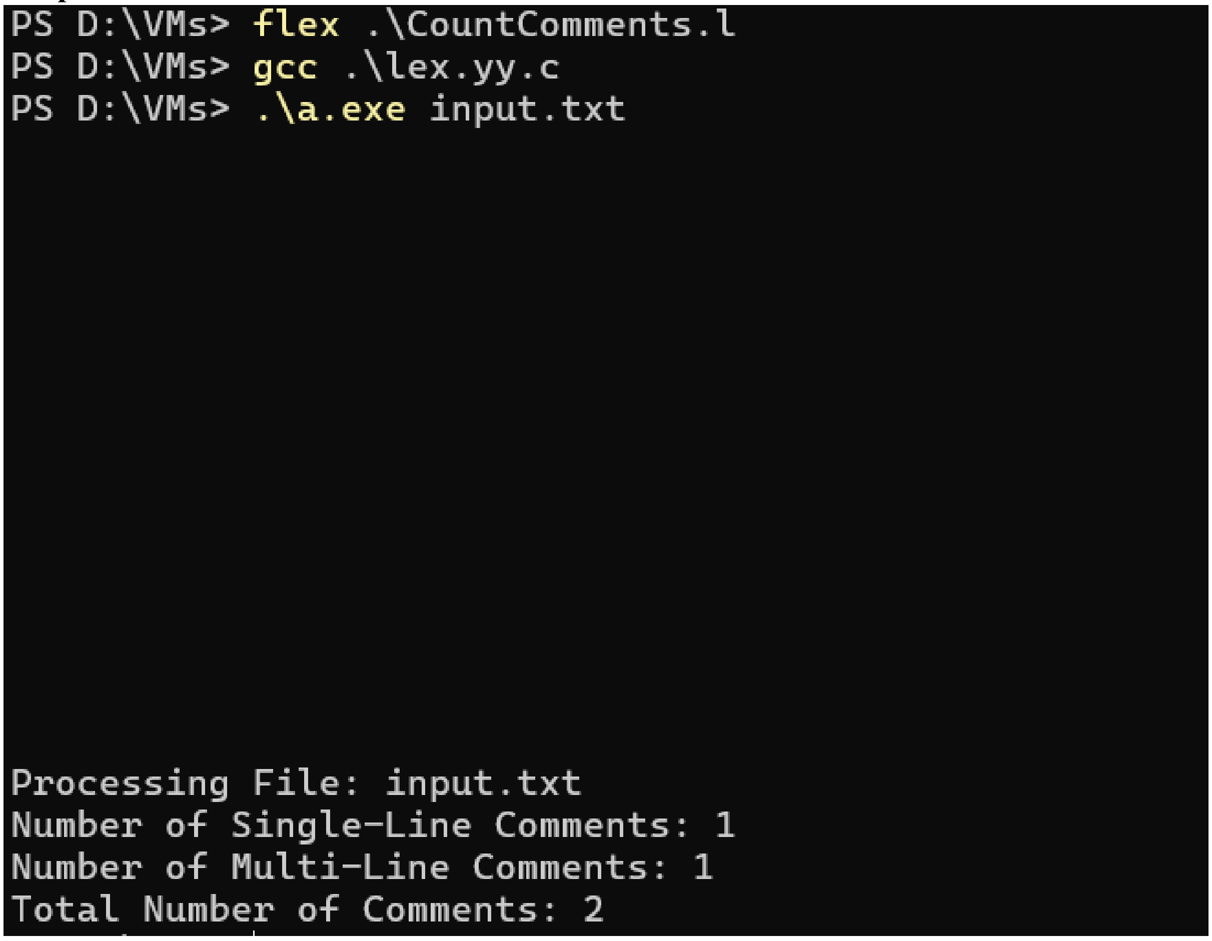
Input.txt

```
#include <stdio.h>

int main() {
    int a, b, sum = 0;

     // Read two numbers from the user
    printf("Enter two integers: ");
    scanf("%d %d", &a, &b);

    /* Calculate the addition of a and b
     using '+' operator*/
    sum = a + b;

    printf("Sum: %d", sum);

    return 0;
}
```

Code:

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

int single_line_comments = 0;
int multi_line_comments = 0;
%}

%%
\/\/.*  { single_line_comments++; }

\/\*[^*]*\*+([^/*][^*]*\*+)*\/                    {
multi_line_comments++; }

. ;

%%

int yywrap() { return 1; }

int main(int argc, char *argv[]) {
    FILE *fp;

    if (argc < 2) {
        printf("Usage: %s <filename>\n", argv[0]);
        return 1;
    }

    fp = fopen(argv[1], "r");
    if (!fp) {
        printf("Error: Unable to open file %s\n",
argv[1]);
        return 1;
    }

    yyin = fp;
    yylex();
    fclose(fp);
    printf("Processing File: %s\n", argv[1]);
    printf("Number of Single-Line Comments: %d\n", single_line_comments);
    printf("Number of Multi-Line Comments: %d\n", multi_line_comments);
    printf("Total Number of Comments: %d\n", single_line_comments + multi_line_comments);

    return 0;
}
```

Output:



```
PS D:\VMs> flex .\CountComments.l
PS D:\VMs> gcc .\lex.yy.c
PS D:\VMs> .\a.exe input.txt

Processing File: input.txt
Number of Single-Line Comments: 1
Number of Multi-Line Comments: 1
Total Number of Comments: 2
```

Conclusion:

In this experiment, we created a lexical analyzer using FLEX to process C program files. The analyzer counted vowels, consonants, words, characters, and lines, identified keywords, operators, and identifiers, classified even and odd integers, and tracked the occurrences of printf and scanf statements. It also categorized English words according to grammatical rules. This experiment showcased the effectiveness of LEX in automating lexical analysis, emphasizing its importance in compiler design, text processing, and language classification by leveraging regular expressions and pattern matching.