Name: Shreyas Sahare
Roll No. 60

Practical No. 1

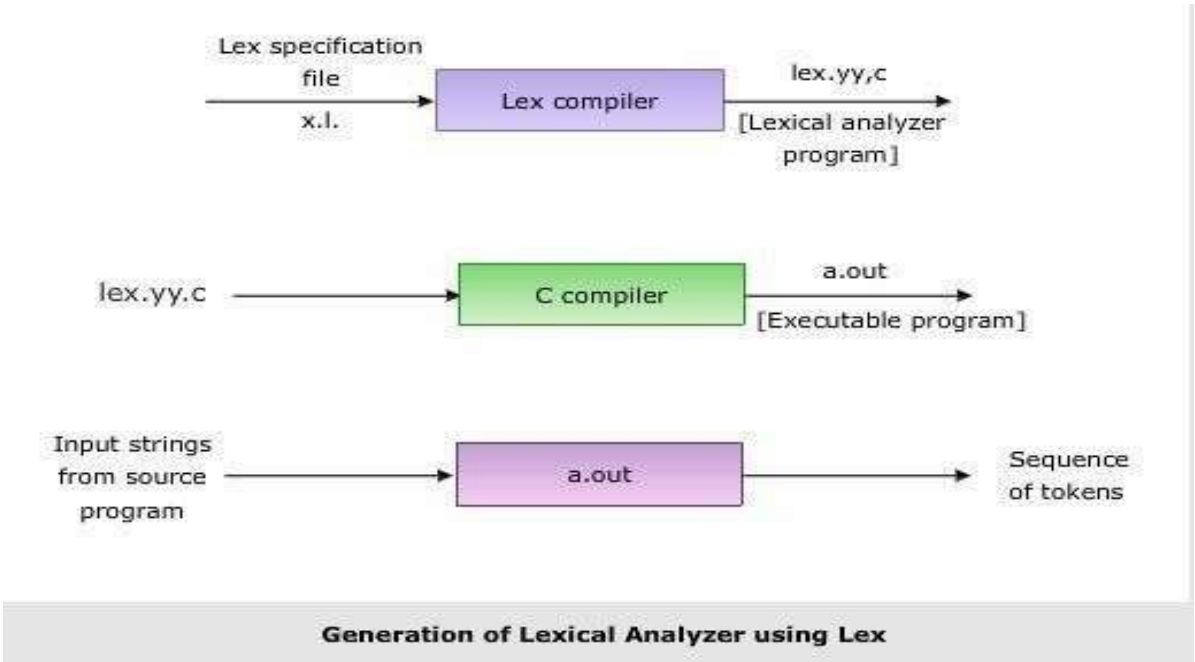
Theory

LEX:
Lex is a program generator designed for lexical processing of character input streams. It accepts a high level, problem-oriented specification for character string matching, and produces a program in a general-purpose language which recognizes regular expressions. The regular expressions are specified by the user in the source specifications given to Lex. The Lex written code recognizes these expressions in an input stream and partitions the input stream into strings matching the expressions. At the boundaries between strings program sections provided by the user are executed. The Lex source file associates the regular expressions and the program fragments. As each expression appears in the input to the program written by Lex, the corresponding fragment is executed.

Lex is not a complete language, but rather a generator representing a new language feature which can be added to different programming languages, called ``host languages." Just as general purpose languages can produce code to run on different com puter hardware, Lex can write code in different host languages.

Lex turns the user's expressions and actions (called source in this pic) into the host general-purpose language; the generated program is named yylex. The yylex program will recognize expressions in a stream (called input in this pic) and perform the specified actions for each expression as it is detected.

Diagram of LEX



Generation of Lexical Analyzer using Lex

Format for Lex file

The general format of Lex source is:

```
{definitions}
%%
{rules}
%%
{user subroutines}
```

where the definitions and the user subroutines are often omitted. The second %% is optional, but the first is required to mark the beginning of the rules. The absolute minimum Lex program is thus %% (no definitions, no rules) which translates into a program which copies the input to the output unchanged.

Regular Expression

A regular expression (or RE) specifies a set of strings that matches it; the functions in this module let you check if a particular string matches a given regular expression (or if a given regular expression matches a particular string, which comes down to the same thing).

Regular expressions can be concatenated to form new regular expressions; if A and B are both regular expressions, then AB is also a regular expression. In general, if a string p matches A and another string q matches B, the string pqwill match AB. This holds unless A or B contain low precedence operations; boundary conditions between A and B; or have numbered group references. Thus, complex expressions can easily be constructed from simpler primitive expressions. Regular expressions can contain both special and ordinary characters. Most ordinary characters, like "A", "a", or "0", are the simplest regular expressions; they simply match themselves. You can concatenate ordinary characters, so last matches the string 'last'. (In the rest of this section, we'll write RE's in this special style, usually without quotes, and strings to be matched 'in single quotes'.)

Some characters, like "|" or "(", are special. Special characters either stand for classes of ordinary characters or affect how the regular expressions around them are interpreted.

Lex Library Routines

Lex library routines are those functions which have a detailed knowledge of the lex functionalities and which can be called to implement various tasks in a lex program.

The following table gives a list of some of the lex routines.

Lex Routine	Description
Main()	Invokes the lexical analyzer by calling the yylex subroutine.
yywrap()	Returns the value 1 when the end of input occurs.
yymore()	Appends the next matched string to the current value of the yytext array rather than replacing the contents of the yytext array.
yyless(int n)	Retains n initial characters in the yytext array and returns the remaining characters to the input stream.

yyreject	Allows the lexical analyzer to match multiple rules for the same input string. (The yyreject subroutine is called when the special action REJECT is used.)
yylex()	The default main() contains the call of yylex()

Answer the Questions:

1. Use of yywrap:
   he `yywrap` function is used in Lex to indicate the end of input. By default, when the input is exhausted (i.e., Lex reaches the end of the input file), it calls `yywrap`. This function returns an integer value, and if it returns 0, Lex expects more input. If it returns 1, Lex terminates the scanning process.

2. Use of yylex function
   The `yylex()` function is a key component of any Lex program. It is automatically generated by Lex when you compile your `.l` file and serves as the main function that performs lexical analysis on the input data. The `yylex()` function scans the input and matches patterns defined in the Lex file, executing the corresponding actions when patterns are matched.

3. What does lex.yy.c. do?

The `lex.yy.c` file is responsible for performing lexical analysis. It contains the following key components:

1. **`yylex()` Function:** This function is the heart of the lexical analyzer. It performs the actual scanning of the input, using the regular expressions defined in your `.l` file, and calls the corresponding action blocks whenever a match is found. The logic for token matching and processing is implemented in this function.
2. **Pattern Matching Code:** The file contains C code to match the patterns you defined in the Lex file. These are typically implemented using a state machine, where the input stream is processed character by character.
3. **`yywrap()` Function:** This function indicates the end of the input. It is defined in the `lex.yy.c` file or needs to be defined in your own Lex file. By default, Lex defines a basic version of this function.

Practical No. E1

Aim : Write a lex code to identify the tokens such as keywords, identifiers, operators, constants (Int,float & character), special symbols and strings for C language using LEX. Use File for the input.

Program:

```
%{
#include<stdio.h>
%}
%%
"int"|"float"|"char"|"for"|"if"|"else"|"do"|"while" printf("%s is keyword" , yytext) ;
[0-9]+ printf("%s is Integer Constant" , yytext) ;
[0-9]+\.[0-9]+ printf("%s is a float constant\n", yytext);
\'[A-zA-Z][a-zA-Z0-9]* printf("&s is identifier\n", yytext);
[=|+|-|*|/|%] printf("%s is an operator\n", yytext);
[,|;|:|{|}|(|)] printf("%s is a special symbol\n", yytext);
\".*\" printf("%s is a string\n", yytext);
%%

int yywrap(){
return 1 ;

}

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

Output:

```
C:\CompilerDesign>flex question1.l

C:\CompilerDesign>gcc lex.yy.c

C:\CompilerDesign>a.exe
#include <stdio.h>

int is keyword main( is a special symbol
) is a special symbol
 { is a special symbol

    int is keyword a = is an operator
 10 is Integer Constant; is a special symbol

    float is keyword b = is an operator
 2.5 is a float constant
; is a special symbol

    float is keyword c; is a special symbol
  / is an operator
/ is an operator
 Change `c` to float is keyword to store the result of `a * is an operator
 b`

    print is keywordf( is a special symbol
"Enter values of A and B: " is a string
) is a special symbol
; is a special symbol

    scanf( is a special symbol
"%d%f" is a string
, is a special symbol
 &a, is a special symbol
 &b) is a special symbol
; is a special symbol
  / is an operator
 Use % is an operator
f for is keyword float is keyword input

    c = is an operator
 a * is an operator
 b; is a special symbol
  / is an operator
/ is an operator
 Perfor is keywordm the multiplication

    print is keywordf( is a special symbol
"Value of C is %.2f\n" is a string
, is a special symbol
 c) is a special symbol
; is a special symbol
  / is an operator
/ is an operator
 Print is keyword the result as a float is keyword

    return 0 is Integer Constant; is a special symbol

} is a special symbol
```

Practical No. E2

Aim: Write a Lex program to find the parameters given below. Consider as input a question paper of
an examination.
1. Count the number of questions.
2. Number of questions that have sub-part and how many donot.
3. Count the total marks.
4. Date of examination
5. Semester
6. Count different types of questions- Eg: What, Discuss, etc.
7. Numbers of words, lines, small letters, capital letters, digits, and special characters.

Program:

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

int question_count = 0;
int subpart_count = 0;
int no_subpart_count = 0;
int total_marks = 0;
char exam_date[20] = "";
char semester[20] = "";
int what_count = 0;
int discuss_count = 0;
int total_words = 0;
int total_lines = 0;
int lowercase_count = 0;
int uppercase_count = 0;
int digit_count = 0;
int special_char_count = 0;

void count_chars(const char *text) {
    for (const char *p = text; *p; p++) {
        if (*p >= 'a' && *p <= 'z') lowercase_count++;
        else if (*p >= 'A' && *p <= 'Z') uppercase_count++;
        else if (*p >= '0' && *p <= '9') digit_count++;
        else special_char_count++;
    }
}
%}
```

```
%%

\n                  { total_lines++; }
"Question [0-9]+"      { question_count++; no_subpart_count++; } "sub-part"            {
subpart_count++; }
"marks [0-9]+"         {
                        int marks;
                        sscanf(yytext, "marks %d", &marks);
                        total_marks += marks;
                    }
"Date: [0-9]{2}/[0-9]{2}/[0-9]{4}" { strcpy(exam_date, yytext); }
"Semester: [A-Za-z]+"  { strcpy(semester, yytext); }
"What"                 { what_count++; }
"Discuss"              { discuss_count++; }
[a-zA-Z]+              { total_words++; count_chars(yytext); }
[0-9]+                 { /* Ignore standalone numbers */ }

%%

int yywrap() {
    return 1;
}

int main(int argc, char **argv) {
    if (argc > 1) {
        FILE *file = fopen(argv[1], "r");
        if (!file) {
            fprintf(stderr, "Could not open file %s\n", argv[1]);
            return 1;
        }
        yyin = file;
    }

    yylex();

    printf("Total Questions: %d\n", question_count);
    printf("Questions with Sub-parts: %d\n", subpart_count);
    printf("Questions without Sub-parts: %d\n", question_count - subpart_count);
    printf("Total Marks: %d\n", total_marks);
    printf("Date of Examination: %s\n", exam_date);
    printf("Semester: %s\n", semester);
    printf("Count of 'What' Questions: %d\n", what_count);
    printf("Count of 'Discuss' Questions: %d\n", discuss_count);
    printf("Total Words: %d\n", total_words);
    printf("Total Lines: %d\n", total_lines);
```

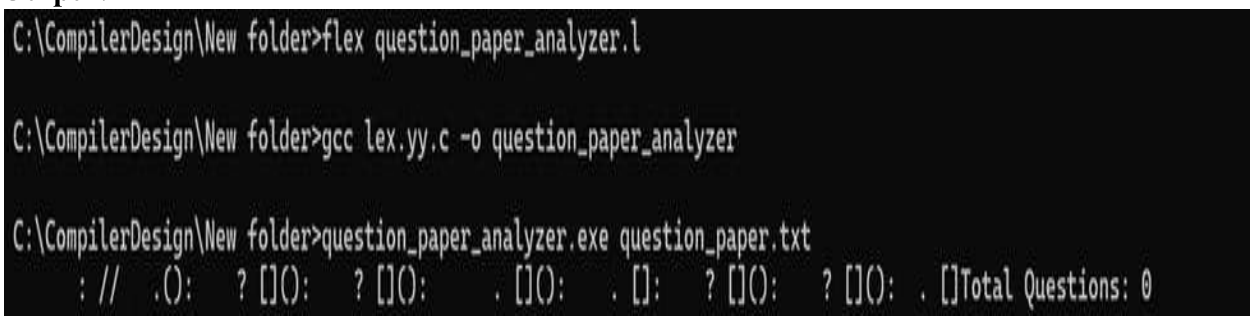
```
    printf("Lowercase Letters: %d\n", lowercase_count);
    printf("Uppercase Letters: %d\n", uppercase_count);
    printf("Digits: %d\n", digit_count);
    printf("Special Characters: %d\n", special_char_count);

    return 0;
}
```

Input:

Shri Ramdeobaba College of Engineering and Management
Sem: VI

12/12/2024
Solve as per questions.
Q1(a): What is a compiler? [5Marks]
Q1(b): What is a software? [5Marks]
Q2(a): Explain the need for an assembler. [3Marks]
Q2(b): Discuss phases of compiler. [2Marks]
Q3: What is a parser? [5Marks]
Q4(a): What is error correction? [2Marks]
Q4(b): Explain SDTS. [3Marks]

Output:



```
Total Questions: 4
Questions with Sub-parts: 3
Questions without Sub-parts: 1
Total Marks: 25
Date of Examination: 12/12/2024
Semester: VI
Count of 'What' Questions: 2
Count of 'Discuss' Questions: 1
Total Words: 38
Total Lines: 13
Lowercase Letters: 138
Uppercase Letters: 26
Digits: 12
Special Characters: 13
```

Practical No. E3

Aim: Write a Lex Program which takes C program from file and write the same C program in another file after removing the comments.

Program:

```
%{
#include<stdio.h>
#include<string.h>
FILE *outfile;
%}

%%
"/*"([^*]|\*+[^*/])*"*"+"/"    ;
"//".*                         ;
.|\n    { fputc(yytext[0], outfile); }
%%

int main(int argc, char *argv[]) {
    if (argc < 3) {
        printf("Usage: %s <input_file> <output_file>\n", argv[0]);
        return 1;
    }

    FILE *infile = fopen(argv[1], "r");
    if (infile == NULL) {
        perror("Error opening input file");
```

```
        return 1;
    }

    outfile = fopen(argv[2], "w");
    if (outfile == NULL) {
        perror("Error opening output file");
        fclose(infile);
        return 1;
    }

    yyin = infile;
    yylex();

    fclose(infile);
    fclose(outfile);

    printf("Comments removed and output written to %s\n", argv[2]);
    return 0;
}

int yywrap() {
    return 1;
}
```

Input:

```
#include<stdio.h>
// This is a single-line comment
int main() {
    /* This is a
       multi-line comment */
    printf("Hello, World!\n");
    return 0; // End of program
}
```

Output:

```
#include<stdio.h>

int main() {

    printf("Hello, World!\n");
    return 0;
}
```

```
C:\CompilerDesign>flex question3.l

C:\CompilerDesign>gcc lex.yy.c

C:\CompilerDesign>a.exe input3.c output3.c
Comments removed and output written to output3.c
```

Practical No. E4

Aim: Write a LEX specification to take the contents from a file
1. Add 3 to number divisible by 7
2. Add 4 to number divisible by 2
3. Convert the alphabetical list to numbered list

Program:

```
%{
#include<stdio.h>
#include<stdlib.h>
FILE *outfile;
int line_number = 1;
%}

%%

[0-9]+ {
    int num = atoi(yytext);r
    if (num % 7 == 0) {
        num += 3;
    } else if (num % 2 == 0) {
        num += 4;
    }
    fprintf(outfile, "%d\n", num);
}

[a-zA-Z]\) {
    fprintf(outfile, "%d) ", line_number++);
}

.* {
    fprintf(outfile, "%s", yytext);
}
```

```
\n {
    fprintf(outfile, "\n");
}
%%

int main(int argc, char *argv[]) {
    if (argc < 3) {
        printf("Usage: %s <input_file> <output_file>\n", argv[0]);
        return 1;
    }

    FILE *infile = fopen(argv[1], "r");
    if (infile == NULL) {
        perror("Error opening input file");
        return 1;
    }

    outfile = fopen(argv[2], "w");
    if (outfile == NULL) {
        perror("Error opening output file");
        fclose(infile);
        return 1;
    }

    yyin = infile;
    yylex();

    fclose(infile);
    fclose(outfile);

    printf("Processed input and written output to %s\n", argv[2]);
    return 0;
}

int yywrap() {
    return 1;
}
```

Input:

```
14
16
a) Bread
b) Butter
```

c) Magnets
d) Paint Brush

Output:
17

20

a) Bread
b) Butter
c) Magnets
d) Paint Brush

```
C:\CompilerDesign>flex question4.l

C:\CompilerDesign>gcc lex.yy.c

C:\CompilerDesign>a.exe input4.txt output4.txt
Processed input and written output to output4.txt
```