

**LAB MANUAL**  
of  
**COMPILER DESIGN LABORATORY**  
**(CSE605)**  
**Bachelor of Technology (CSE)**  
By

Om Pandit (22000658)

Third Year, Semester 6



Department of Computer Science and Engineering  
School Engineering and Technology  
Navrachana University, Vadodara  
Spring Semester  
(2024-2025)

# List of Experiments

Sr. No.	Name of Experiment	Page No.
1	<p><b>A.</b> Write a program to recognise strings starts with ‘a’ over {a, b}.</p> <p><b>B.</b> Write a program to recognise strings end with ‘a’.</p> <p><b>C.</b> Write a program to recognise strings end with ‘ab’. Take the input from text file.</p> <p><b>D.</b> Write a program to recognise strings contains ‘ab’. Take the input from text file.</p>	3
2	<p><b>A.</b> Write a program to recognise the valid identifiers.</p> <p><b>B.</b> Write a program to recognise the valid operators.</p> <p><b>C.</b> Write a program to recognise the valid number.</p> <p><b>D.</b> Write a program to recognise the valid comments.</p> <p><b>E.</b> Program to implement Lexical Analyser.</p>	10
3	To Study about Lexical Analyzer Generator (LEX) and Flex(Fast Lexical Analyzer).	21
4	<p>Implement following programs using Lex :</p> <p><b>A.</b> Write a Lex program to take input from text file and count no of characters, no. of lines &amp; no. of words.</p> <p><b>B.</b> Write a Lex program to take input from text file and count number of vowels and consonants.</p> <p><b>C.</b> Write a Lex program to print out all numbers from the given file.</p> <p><b>D.</b> Write a Lex program which adds line numbers to the given file and display the same into different file.</p> <p><b>E.</b> Write a Lex program to printout all markup tags and HTML comments in file.</p>	23
5	<p><b>A.</b> Write a Lex program to count the number of C comment lines from a given C program. Also eliminate them and copy that program into separate file.</p> <p><b>B.</b> Write a Lex program to recognise keywords, identifiers, operators, numbers, special symbols, literals from a given C program.</p>	33
6	Program to implement Recursive Descent Parsing in C.	39
7	<p>To Study about <b>Yet Another Compiler - Compiler (YACC)</b> :</p> <p><b>A.</b> Create Yacc and Lex specification files to recognises arithmetic expressions involving +, -, * and / .</p> <p><b>B.</b> Create Yacc and Lex specification files are used to generate a calculator which accepts integer type arguments.</p> <p><b>C.</b> Create Yacc and Lex specification files are used to convert infix expression to postfix expression.</p>	41

## Experiment - 1

**Aim :** Write a C program for the following :

**A.** Write a program to recognise strings starts with 'a' over {a, b}.

**Code :**

```
#include <stdio.h>

int main(){
    char input[10];
    int i=0, state=0;
    printf("INPUT THE STRING : ");
    scanf("%s", input);

    while(input[i]!='\0'){
        switch(state) {
            case 0:
                if(input[i]=='a') state = 1;
                else if (input[i] == 'b') state = 2;
                else state = 3;
                break;

            case 1:
                if (input[i] == 'a' || input[i] == 'b') state = 1;
                else state = 3;
                break;

            case 2:
                if (input[i] == 'a' || input[i] == 'b') state = 2;
                else state = 3;
                break;

            case 3:
                state = 3;
                break;
        }

        i++;
    }

    if (state==1) printf("STRING IS VALID!");
    else if (state==2 || state==0) {printf("STRING IS INVALID!");}
    else if (state==3) printf("STRING IS NOT RECOGNISED!");
    return 0;
}
```

**Output :**

```
Last login: Mon Jan 27 19:39:36 on ttys001
/Users/ompandit/Desktop/SEM\ 6/COMPILER\ DESIGN/P1-\(a\)\ ; exit;
ompandit@Oms-MacBook-Pro ~ % /Users/ompandit/Desktop/SEM\ 6/COMPILER\ DESIGN/P1-\(a\)\ ; exit;
INPUT THE STRING : aaaa aba bbb baa
STRING IS VALID!
Saving session...
...copying shared history...
...saving history...truncating history files...
...completed.

[Process completed]()
```

**B.** Write a program to recognise strings end with ‘a’.

**Code :**

```
#include <stdio.h>

int main(){
    char input[10];
    int i=0, state=0;
    printf("INPUT THE STRING : ");
    scanf("%s", input);

    while(input[i]!='\0'){
        switch(state) {
            case 0:
                if(input[i]=='a') state = 1;
                else state = 0;
                break;

            case 1:
                if (input[i] == 'a' || input[i] == 'b') state = 0;
                else state = 0;
                break;

            case 2:
                if (input[i] == 'a' || input[i] == 'b') state = 1;
                else state = 1;
                break;
        }

        i++;
    }

    if (state==1) printf("STRING IS VALID!");
    else if (state==2 || state==0) {printf("STRING IS INVALID!");}
    else if (state==3) printf("STRING IS NOT RECOGNISED!");
    return 0;
}
```

**Output :**

```
Last login: Mon Jan 27 19:39:50 on ttys001
/Users/ompandit/Desktop/SEM\ 6/COMPILER\ DESIGN/P1-\(b\)\ ; exit;
ompandit@Oms-MacBook-Pro ~ % /Users/ompandit/Desktop/SEM\ 6/COMPILER\ DESIGN/P1-\(b\)\ ; exit;
INPUT THE STRING : aba bbb bab aaaa
STRING IS VALID!
Saving session...
...copying shared history...
...saving history...truncating history files...
...completed.

[Process completed]
```

C. Write a program to recognise strings end with 'ab'. Take the input from text file.

**Code :**

```
#include <stdio.h>
#include <string.h>

int ends_with_ab(const char *str) {
    int len = strlen(str);

    if (len > 0 && str[len - 1] == '\n') {
        len--;
    }

    if (len >= 2 && str[len - 2] == 'a' && str[len - 1] == 'b') {
        return 1;
    }
    return 0;
}

int main() {
    FILE *file = fopen("input.txt", "r");
    if (!file) {
        perror("CANNOT OPEN INPUT FILE!");
        return 1;
    }

    char line[256];
    while (fgets(line, sizeof(line), file)) {
        if (ends_with_ab(line)) {
            printf("STRING ENDS WITH 'ab' : %s", line);
        }
    }

    fclose(file);
    return 0;
}
```

**Output :**

```
C P1-(c).c      ls
ls
input.txt
1 helloab
2 thisisatest
3 grab
4 abcab
5 about
6 tab
7 cab
8 labcoat
9 ab
10 endwithab
```

```
=thread-selected,id="1"
STRING ENDS WITH 'ab' : helloab
STRING ENDS WITH 'ab' : grab
STRING ENDS WITH 'ab' : abcab
STRING ENDS WITH 'ab' : tab
STRING ENDS WITH 'ab' : cab
STRING ENDS WITH 'ab' : ab
STRING ENDS WITH 'ab' : endwithab
The program '/Users/ompandit/Desktop/SEM 6/COMPILER DESIGN/P1-(c)' has exited with code 0 (0x00000000).
```

D. Write a program to recognise strings contains 'ab'. Take the input from text file.

Code :

```
#include <stdio.h>
#include <string.h>

int main() {
    FILE *file;
    char input[100];
    int i, state;

    file = fopen("input.txt", "r");
    if (file == NULL) {
        printf("ERROR : FILE COULD NOT BE FOUND!\n");
        return 1;
    }

    while (fgets(input, sizeof(input), file)) {
        input[strcspn(input, "\n")] = 0;
        i = 0;
        state = 0;

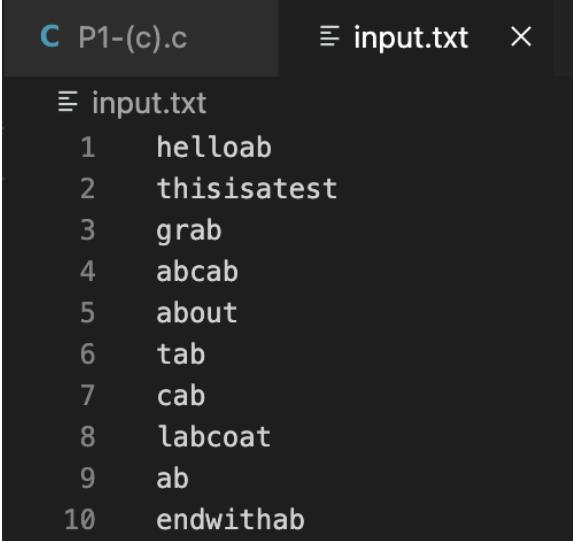
        while (input[i] != '\0') {
            switch (state) {
                case 0:
                    if (input[i] == 'a') state = 1;
                    else state = 0;
                    break;

                case 1:
                    if (input[i] == 'b') state = 2;
                    else state = 0;
                    break;

                case 2:
                    state = 2;
                    break;
            }
            i++;
        }

        if (state == 2) printf("THE STRING \"%s\" CONTAINS 'ab'.\n", input);
        else printf("THE STRING \"%s\" DOES NOT CONTAIN 'ab'.\n", input);
    }

    fclose(file);
    return 0;
}
```

**Output :**

```
C P1-(c).c      ≡ input.txt  ×

≡ input.txt
1  helloab
2  thisisatest
3  grab
4  abcab
5  about
6  tab
7  cab
8  labcoat
9  ab
10 endwithab
```

```
=thread-selected,id="1"
THE STRING "helloab" CONTAINS 'ab'.
THE STRING "thisisatest" DOES NOT CONTAIN 'ab'.
THE STRING "grab" CONTAINS 'ab'.
THE STRING "abcab" CONTAINS 'ab'.
THE STRING "about" CONTAINS 'ab'.
THE STRING "tab" CONTAINS 'ab'.
THE STRING "cab" CONTAINS 'ab'.
THE STRING "labcoat" CONTAINS 'ab'.
THE STRING "ab" CONTAINS 'ab'.
THE STRING "endwithab" CONTAINS 'ab'.
THE STRING "" DOES NOT CONTAIN 'ab'.
The program '/Users/ompandit/Desktop/SEM 6/COMPILER DESIGN/P1-(d)' has exited with code 0 (0x00000000).
```

## Experiment - 2

**Aim :** Write a Python program for the following :

A. Write a program to recognise the valid identifiers.

**Code :**

```

def is_valid_identifier(token):
    state = 0
    for char in token:
        if state == 0:
            if char.isalpha() or char == '_':
                state = 1
            else:
                return False
        elif state == 1:
            if char.isalnum() or char == '_':
                state = 1
            else:
                return False
    return state == 1

def is_keyword(token):
    keywords = {"if", "else", "while", "return", "for", "def",
    "class", "import", "from", "as", "with", "try", "except",
    "finally", "raise", "lambda", "pass", "break", "continue", "in",
    "not", "or", "and", "is", "None", "True", "False", "global",
    "nonlocal", "assert", "yield"}
    return token in keywords

def tokenize_and_check(input_string):
    tokens = input_string.split()
    results = []
    for token in tokens:
        identifier = is_valid_identifier(token)
        keyword_check = is_keyword(token)
        status = "Both Identifier and Keyword" if identifier and
keyword_check else \
            "Valid Identifier" if identifier else \
            "Keyword" if keyword_check else "Invalid"
        results.append((token, status))

    return results

if __name__ == "__main__":
    with open("input.txt", "r") as file:
        input_string = file.read().strip()

    results = tokenize_and_check(input_string)

    with open("output.txt", "w") as file:
        file.write("Tokenized Output:\n")

```

```
for token, status in results:
    file.write(f"Token: '{token}', Status: {status}\n")
```

```
print("\nTokenized Output:")
for token, status in results:
    print(f"Token: '{token}', Status: {status}")
```

**Output :**

```
≡ input.txt
1  if
2  else
3  my_var
4  lvariable
5  _underscore
6  class
7  def
8  function_name
9  99bottles
10 helloworld
11 int
12 float
13 __init__
14 try
15 except
16 lambda
17 whileTrue
18 True
19 False
20 None
```

PROBLEMS    OUTPUT    DEBUG CONSOLE    TERMINAL

```
/opt/homebrew/bin/python3 "/Users/ompandit/Desktop/SEM 6/COMPILER DESIGN/P2-(a).py"
● ompandit@Oms-MacBook-Pro COMPILER DESIGN % /opt/homebrew/bin/python3 "/Users/ompandit/Desktop/SEM 6/COMPILER DESIGN/P2-(a).py"

Tokenized Output:
Token: 'if', Status: Both Identifier and Keyword
Token: 'else', Status: Both Identifier and Keyword
Token: 'my_var', Status: Valid Identifier
Token: 'lvariable', Status: Invalid
Token: '_underscore', Status: Valid Identifier
Token: 'class', Status: Both Identifier and Keyword
Token: 'def', Status: Both Identifier and Keyword
Token: 'function_name', Status: Valid Identifier
Token: '99bottles', Status: Invalid
Token: 'helloworld', Status: Valid Identifier
Token: 'int', Status: Valid Identifier
Token: 'float', Status: Valid Identifier
Token: '__init__', Status: Valid Identifier
Token: 'try', Status: Both Identifier and Keyword
Token: 'except', Status: Both Identifier and Keyword
Token: 'lambda', Status: Both Identifier and Keyword
Token: 'whileTrue', Status: Valid Identifier
Token: 'True', Status: Both Identifier and Keyword
Token: 'False', Status: Both Identifier and Keyword
Token: 'None', Status: Both Identifier and Keyword
○ ompandit@Oms-MacBook-Pro COMPILER DESIGN %
```

**B.** Write a program to recognise the valid operators.

**Code :**

```
with open("input2_b.txt", "r") as file:
    input_str = file.read().split()

operators = {"+", "-", "%", "*", "/", "<", ">", "--", "!=" , "+="}
i = 0

tokens = []

while i < len(input_str):
    for char in input_str:
        if char in operators:
            print(f"{char} IS AN OPERATOR")
            tokens.append((char, "operator"))
        else:
            print(f"{char} IS AN IDENTIFIER")
            tokens.append((char, "identifier"))
    i += 1
```

**Output :**

```
≡ input2_b.txt
1    a + b = c
2
```

```
PROBLEMS    OUTPUT    DEBUG CONSOLE    TERMINAL
/opt/homebrew/bin/python3 "/Users/ompandit/Desktop/SEM 6/COMPILER DESIGN/P2-(b).py"
● ompandit@Oms-MacBook-Pro COMPILER DESIGN % /opt/homebrew/bin/python3 "/Users/ompandit/Desktop/SEM 6/COMPILER DESIGN/P2-(b).py"
a IS AN IDENTIFIER
+ IS AN OPERATOR
= IS AN IDENTIFIER
c IS AN IDENTIFIER
1 IS AN IDENTIFIER
2 IS AN IDENTIFIER
● ompandit@Oms-MacBook-Pro COMPILER DESIGN %
```

C. Write a program to recognise the valid number.

Code :

```
def is_valid_number_fsm(number: str) -> bool:
    state = 'a'
    for char in number:
        if state == 'a':
            if char in '+-':
                state = 'h'
            elif char.isdigit():
                state = 'b'
            else:
                return False
        elif state == 'h':
            if char.isdigit():
                state = 'b'
            else:
                return False
        elif state == 'b':
            if char.isdigit():
                state = 'b'
            elif char == '.':
                state = 'c'
            elif char in 'Ee':
                state = 'e'
            else:
                return False
        elif state == 'c':
            if char.isdigit():
                state = 'd'
            else:
                return False
        elif state == 'd':
            if char.isdigit():
                state = 'd'
            elif char in 'Ee':
                state = 'e'
            else:
                return False
        elif state == 'e':
            if char in '+-':
                state = 'f'
            elif char.isdigit():
                state = 'g'
            else:
                return False
        elif state == 'f':
            if char.isdigit():
                state = 'g'
            else:
                return False
        else:
            return False
```

```

        elif state == 'g':
            if char.isdigit():
                state = 'g'
            else:
                return False
        return state in {'b', 'd', 'g'}
```

```

if __name__ == "__main__":
    try:
        with open("numbers.txt", "r") as file:
            for line in file:
                number = line.strip()
                print(f"'{number}' IS A VALID NUMBER : {is_valid_number_fsm(number)}")
    except FileNotFoundError:
        print("ERROR : 'numbers.txt' FILE NOT FOUND.")
```

**Output :**

```

≡ numbers.txt
1 123
2 -456.78
3 3.14159
4 1E10
5 -2.5e-3
6 +100
7 abc
8 12.34.56
9 E45
10 1.2.3
```

```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL
/opt/homebrew/bin/python3 "/Users/ompandit/Desktop/SEM 6/COMPILER DESIGN/P3-(c).py"
● ompandit@Oms-MacBook-Pro COMPILER DESIGN % /opt/homebrew/bin/python3 "/Users/ompandit/Desktop/SEM 6/COMPILER DESIGN/P3-(c).py"
'123' IS A VALID NUMBER : True
'-456.78' IS A VALID NUMBER : True
'3.14159' IS A VALID NUMBER : True
'1E10' IS A VALID NUMBER : True
'-2.5e-3' IS A VALID NUMBER : True
'+100' IS A VALID NUMBER : True
'abc' IS A VALID NUMBER : False
'12.34.56' IS A VALID NUMBER : False
'E45' IS A VALID NUMBER : False
'1.2.3' IS A VALID NUMBER : False
' ' IS A VALID NUMBER : False
'.' IS A VALID NUMBER : False
○ ompandit@Oms-MacBook-Pro COMPILER DESIGN %
```

D. Write a program to recognise the valid comments.

Code :

```
def is_valid_comment(line: str) -> bool:
    state = 'start'
    i = 0
    while i < len(line):
        char = line[i]

        if state == 'start':
            if char == '/':
                state = 'slash'
            else:
                return False

        elif state == 'slash':
            if char == '/':
                return True
            elif char == '*':
                state = 'multi_line'
            else:
                return False

        elif state == 'multi_line':
            if char == '*':
                state = 'multi_line_end'
            else:
                state = 'multi_line'

        elif state == 'multi_line_end':
            if char == '/':
                return True
            elif char != '*':
                state = 'multi_line'

        i += 1

    return state == 'multi_line_end'

if __name__ == "__main__":
    try:
        with open("comments.txt", "r") as file:
            for line in file:
                line = line.strip()
                print(f"'{line}' is a valid comment:
{is_valid_comment(line)}")
    except FileNotFoundError:
        print("Error: 'comments.txt' file not found.")
```

**Output :**

```
≡ comments.txt
1  Hello World
2  // This is a single-line comment
3  /* This is a multi-line comment */
4  Not a comment
5  /* Unclosed comment
```

```
PROBLEMS    OUTPUT    DEBUG CONSOLE    TERMINAL
/opt/homebrew/bin/python3 "/Users/ompandit/Desktop/SEM 6/COMPILER DESIGN/P2-(d).py"
● ompandit@Oms-MacBook-Pro COMPILER DESIGN % /opt/homebrew/bin/python3 "/Users/ompandit/Desktop/SEM 6/COMPILER DESIGN/P2-(d).py"
'Hello World' is a valid comment: False
'// This is a single-line comment' is a valid comment: True
'/* This is a multi-line comment */' is a valid comment: True
'Not a comment' is a valid comment: False
'/* Unclosed comment' is a valid comment: False
○ ompandit@Oms-MacBook-Pro COMPILER DESIGN %
```

## E. Program to implement Lexical Analyser.

Code :

```

def check(lexeme):
    keywords = {"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"}
    if lexeme in keywords:
        print(f"{lexeme} is a keyword")
    else:
        print(f"{lexeme} is an identifier")

def lexer(filename):
    try:
        with open(filename, "r") as f:
            buffer = f.read()
    except FileNotFoundError:
        print("Error opening file")
    return buffer

state = 0
lexeme = ""
f = 0
while f < len(buffer):
    c = buffer[f]
    if state == 0:
        if c.isalpha() or c == '_':
            state = 1
            lexeme += c
        elif c.isdigit():
            state = 13
            lexeme += c
        elif c == '/':
            state = 11
        elif c in "\t\n":
            state = 0
        elif c in ";,+*/%=<>(){}[]":
            print(f"{c} is a symbol")
            state = 0
        else:
            state = 0
    elif state == 1:
        if c.isalnum() or c == '_':
            lexeme += c
        else:
            check(lexeme)
    f += 1

```

```

        lexeme = ""
        state = 0
        f -= 1
    elif state == 11:
        if c == '/':
            while f < len(buffer) and buffer[f] != '\n':
                f += 1
            state = 0
        elif c == '*':
            f += 1
            while f < len(buffer) - 1 and not (buffer[f] == '*' and buffer[f + 1] == '/'):
                f += 1
            f += 2
            state = 0
        else:
            print("/ is an operator")
            state = 0
            f -= 1
    elif state == 13:
        if c.isdigit():
            lexeme += c
        elif c == '.':
            state = 14
            lexeme += c
        elif c in "Ee":
            state = 16
            lexeme += c
        else:
            print(f"{lexeme} is a valid integer")
            lexeme = ""
            state = 0
            f -= 1
    elif state == 14:
        if c.isdigit():
            lexeme += c
            state = 15
        else:
            print("Error: Invalid floating point format")
            lexeme = ""
            state = 0
    elif state == 15:
        if c.isdigit():
            lexeme += c
        elif c in "Ee":
            state = 16
            lexeme += c
        else:
            print(f"{lexeme} is a valid floating point
number")
            lexeme = ""
            state = 0

```

```

        f -= 1
    elif state == 16:
        if c in "+-":
            state = 17
            lexeme += c
        elif c.isdigit():
            state = 18
            lexeme += c
        else:
            print("Error: Invalid scientific notation")
            lexeme = ""
            state = 0
    elif state == 17:
        if c.isdigit():
            state = 18
            lexeme += c
        else:
            print("Error: Invalid exponent format")
            lexeme = ""
            state = 0
    elif state == 18:
        if c.isdigit():
            lexeme += c
        else:
            print(f"{lexeme} is a valid scientific notation
number")
            lexeme = ""
            state = 0
            f -= 1
    f += 1

lexer("input2.txt")

```

**Output :**

```

≡ input2.txt
1 // This is a single-line comment
2 /* This is
3 | a multi-line comment */
4 int main() {
5     int a = 10;
6     float b = 3.14;
7     char c = 'A';
8     if (a < b) {
9         a = a + 1;
10    }
11    return 0;
12 }

```

PROBLEMS    OUTPUT    DEBUG CONSOLE    TERMINAL

```
/opt/homebrew/bin/python3 "/Users/ompandit/Desktop/SEM 6/COMPILER DESIGN/P2-(e).py"
● ompandit@Oms-MacBook-Pro COMPILER DESIGN % /opt/homebrew/bin/python3 "/Users/ompandit/Desktop/SEM 6/COMPILER DESIGN/P2-(e).py"
int is a keyword
main is an identifier
( is a symbol
) is a symbol
{ is a symbol
int is a keyword
a is an identifier
= is a symbol
10 is a valid integer
; is a symbol
float is a keyword
b is an identifier
= is a symbol
3.14 is a valid floating point number
; is a symbol
char is a keyword
c is an identifier
= is a symbol
A is an identifier
; is a symbol
if is a keyword
( is a symbol
a is an identifier
< is a symbol
b is an identifier
) is a symbol
{ is a symbol
a is an identifier
= is a symbol
a is an identifier
+ is a symbol
1 is a valid integer
; is a symbol
} is a symbol
return is a keyword
0 is a valid integer
; is a symbol
} is a symbol
○ ompandit@Oms-MacBook-Pro COMPILER DESIGN %
```

## Experiment - 3

**Aim :** To Study about Lexical Analyzer Generator (LEX) and Flex(Fast Lexical Analyzer).

### DESCRIPTION :

Lexical analysis is the first phase of a compiler, responsible for converting source code into tokens. This phase is automated using Lexical Analyzer Generators like LEX and Flex.

### LEX (Lexical Analyzer Generator)

LEX is a tool used for generating lexical analyzers in compiler design. It helps in pattern recognition and tokenising input text using regular expressions. LEX works by defining patterns and corresponding actions in a .l file, which is then processed to generate a C-based scanner.

#### Key Features of LEX :

- Uses regular expressions to match patterns in input text.
- Generates lex.yy.c, a C program implementing the scanner.
- Can be compiled using a C compiler to produce an executable lexer.
- Works with YACC (Yet Another Compiler Compiler) to build full-fledged compilers.

#### Working of LEX :

- Specification: The user writes a .l file containing regular expressions and C actions.
- Processing: The lex command processes the .l file and generates lex.yy.c.
- Compilation: The lex.yy.c is compiled with gcc to create an executable scanner.
- Execution: The scanner reads input, matches patterns, and executes the corresponding actions.

### Flex (Fast Lexical Analyzer)

Flex is an enhanced and faster version of LEX, designed for improved performance and portability. It follows the same working mechanism as LEX but generates more efficient and optimized C code.

#### Key Features of Flex :

- Faster and more efficient than LEX.
- Uses longest match rule over first match rule.
- Generates lex.yy.c, similar to LEX but optimized for better performance.
- Works seamlessly on Linux, Unix, and Windows with the required dependencies.

#### Working of Flex :

- Write a `` file with pattern definitions and C-based actions.
- Use the `` command to generate lex.yy.c.
- Compile the file using gcc.
- Run the executable, which scans the input and processes tokens.

## Differences Between LEX and Flex

<b>Feature</b>	<b>LEX</b>	<b>Flex</b>
<b>Speed</b>	Slower	Faster
<b>Portability</b>	Limited	Widely used in Linux & Unix
<b>Memory Usage</b>	Higher	Optimized
<b>Output File</b>	lex.yy.c	lex.yy.c
<b>Default Action</b>	Returns first match	Returns longest match

## Procedure

- Create a .l file (e.g., lexer.l) containing regular expressions and C code.
- Use the flex command to generate lex.yy.c.
- Compile the generated C file using GCC.
- Run the executable and provide input for analysis.

## Example Code (LEX/Flex Program)

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

[0-9]+ { printf("NUMBER\n"); }
[a-zA-Z]+ { printf("IDENTIFIER\n"); }
. { printf("SPECIAL CHARACTER\n"); }

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

## Conclusion

LEX and Flex are powerful tools for lexical analysis in compilers. They help automate tokenisation using regular expressions and C functions, making lexical analysis efficient.

## Experiment - 4

**Aim :** Implement the following using LEX :

- A. Write a Lex program to take input from text file and count no of characters, no. of lines & no. of words.

**Code :**

```
P4 > ≡ input.txt
1 Hello
2 Good Morning
3 This is my lex program
4 123456 677 34.676
5 56e56
```

```
P4 > ≡ sample.l
1 %{ 
2 #include<stdio.h>
3 int char_count=0, word_count=0, line_count=0;
4 %}
5 %%
6 \n {line_count++;word_count++;}
7 [\t ]+ word_count++;
8 . char_count++;
9 %%
10 void main(){
11 yyin=fopen("input.txt","r");
12 yylex();
13 printf("This file contains %d characters\n", char_count);
14 printf("This file contains %d words\n", word_count);
15 printf("This file contains %d lines\n", line_count);
16 }
17 int yywrap(){ return(1);} 
```

**Output :**

Name	Date Modified	Size	Kind
a.out	Today at 3:13PM	53 KB	Document
input.txt	Today at 3:05PM	66 bytes	Plain Text
lex.yy.c	Today at 3:13PM	45 KB	text document
sample.l	Today at 3:10PM	393 bytes	text document

```
[ompandit@Oms-MacBook-Pro P4 % flex sample.l
[ompandit@Oms-MacBook-Pro P4 % gcc lex.yy.c
[ompandit@Oms-MacBook-Pro P4 % ./a.out
This file contains 54 characters
This file contains 12 words
This file contains 5 lines
ompandit@Oms-MacBook-Pro P4 % ]
```

B. Write a Lex program to take input from text file and count number of vowels and consonants.

Code :

```
P4-2 > ≡ input.txt
1  Hello
2  Good Morning
3  This is my lex program
4  123456 677 34.676
5  56e56
```

```
P4-2 > ≡ d2.l
1  %{
2  #include<stdio.h>
3  int consonants=0, vowels=0;
4  %}
5  %%
6  [aeiouAEIOU] {vowels++;}
7  [a-zA-Z] {consonants++;}
8  . ;
9  %%
10 int main(){
11     yyin=fopen("input.txt","r");
12     yylex();
13     printf("This file contains .....");
14     printf("\n\t%d vowels ",vowels);
15     printf("\n\t%d consonants ",consonants);
16     return 0;
17 }
18 int yywrap(){ return(1);}
```

**Output :**

Name	Date Modified	Size	Kind
a.out	Today at 3:23PM	53 KB	Document
d2.l	Today at 3:24PM	313 bytes	text document
input.txt	Today at 3:20PM	67 bytes	Plain Text
lex.yy.c	Today at 3:22PM	44 KB	text document

```
[ompandit@Oms-MacBook-Pro ~ % cd Desktop
[ompandit@Oms-MacBook-Pro Desktop % cd SEM\ 6
[ompandit@Oms-MacBook-Pro SEM 6 % cd COMPILER\ DESIGN
[ompandit@Oms-MacBook-Pro COMPILER DESIGN % cd P4-2
[ompandit@Oms-MacBook-Pro P4-2 % flex d2.l
[ompandit@Oms-MacBook-Pro P4-2 % gcc lex.yy.c
[ompandit@Oms-MacBook-Pro P4-2 % ./a.out
```

This file contains .....

12 vowels

23 consonants %

```
[ompandit@Oms-MacBook-Pro P4-2 %
```

C. Write a Lex program to print out all numbers from the given file.

**Code :**

```
P4-3 > ≡ input.txt
1  Hello
2  Good Morning
3  This is my lex program
4  123456
5  677
6  34.676
7  56e56
```

```
P4-3 > ≡ d3.l
1  %}
2  #include <stdio.h>
3  #include <stdlib.h>
4  %}
5
6  digits      [0-9] +
7
8  %%
9  {digits}(\.{digits})?([eE][+-]?{digits})?      { printf("%s is valid number\n", yytext); }
10 \n
11 .
12 %%
13
14 int main() {
15     yyin = fopen("input.txt", "r");
16     if (!yyin) {
17         perror("Error opening input.txt");
18         return 1;
19     }
20     yylex();
21     fclose(yyin);
22     return 0;
23 }
24
25 int yywrap() {
26     return 1;
27 }
```

**Output :**

Name	Date Modified	Size	Kind
a.out	Today at 3:37PM	53 KB	Document
d3.l	Today at 3:38PM	472 bytes	text document
input.txt	Today at 3:37PM	69 bytes	Plain Text
lex.yy.c	Today at 3:37PM	45 KB	text document

```
[ompandit@Oms-MacBook-Pro ~ % cd Desktop
[ompandit@Oms-MacBook-Pro Desktop % cd SEM\ 6
[ompandit@Oms-MacBook-Pro SEM 6 % cd COMPILER\ DESIGN
[ompandit@Oms-MacBook-Pro COMPILER DESIGN % cd P4-3
[ompandit@Oms-MacBook-Pro P4-3 % flex d3.l
[ompandit@Oms-MacBook-Pro P4-3 % gcc lex.yy.c
[ompandit@Oms-MacBook-Pro P4-3 % ./a.out
123456 is valid number
677 is valid number
34.676 is valid number
56e56 is valid number
ompandit@Oms-MacBook-Pro P4-3 %
```

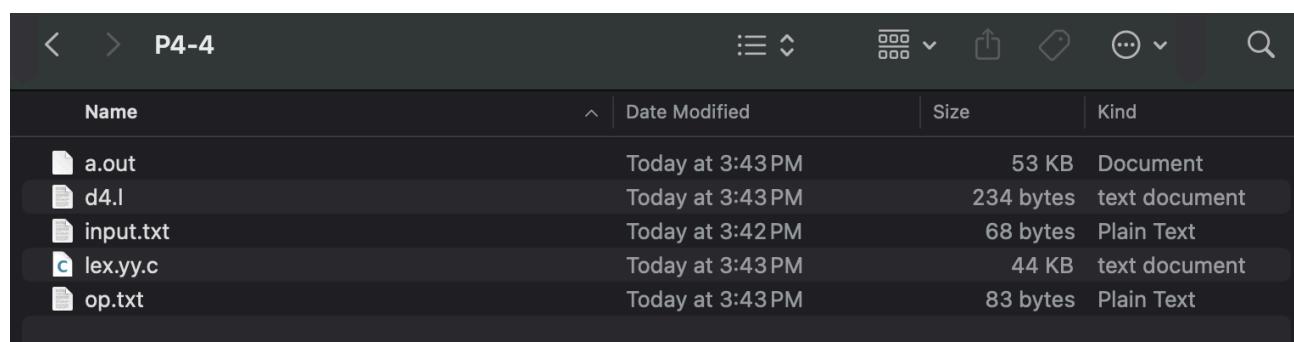
D. Write a Lex program which adds line numbers to the given file and display the same into different file.

**Code :**

```
P4-4 > ≡ input.txt
1 Hello
2 Good Morning
3 This is my lex program
4 123456 677 34.676
5 56e56
```

```
P4-4 > ≡ d4.l
1 %{ 
2     int line_number = 1;
3 }
4 %%
5 .* { fprintf(yyout, "%d: %s", line_number, yytext); line_number++; }
6
7 %%
8     int main(){
9     yyin=fopen("input.txt","r");
10    yyout=fopen("op.txt","w");
11    yylex();
12    printf("done");
13    return 0;
14 }
15    int yywrap(){ return(1);}
```

**Output :**



Name	Date Modified	Size	Kind
a.out	Today at 3:43PM	53 KB	Document
d4.l	Today at 3:43PM	234 bytes	text document
input.txt	Today at 3:42PM	68 bytes	Plain Text
lex.yy.c	Today at 3:43PM	44 KB	text document
op.txt	Today at 3:43PM	83 bytes	Plain Text

```
[ompandit@Oms-MacBook-Pro ~ % cd Desktop
[ompandit@Oms-MacBook-Pro Desktop % cd SEM\ 6
[ompandit@Oms-MacBook-Pro SEM 6 % cd COMPILER\ DESIGN
[ompandit@Oms-MacBook-Pro COMPILER DESIGN % cd P4-4
[ompandit@Oms-MacBook-Pro P4-4 % flex d4.1
[ompandit@Oms-MacBook-Pro P4-4 % gcc lex.yy.c
[ompandit@Oms-MacBook-Pro P4-4 % ./a.out
done%
ompandit@Oms-MacBook-Pro P4-4 % ]
```

P4-4 > ≡ op.txt

1 1: Hello  
2 2: Good Morning  
3 3: This is my lex program  
4 4: 123456 677 34.676  
5 5: 56e56

E. Write a Lex program to printout all markup tags and HTML comments in file.

**Code :**

```
P4-5 > ≡ input.txt
1 <html>
2 <head> Heer </head>
3 <body>
4 <!-- iehhfjs 122 -->
5 </body>
6 </html>
7
```

```
P4-5 > ≡ d5.l
1 %{
2 #include<stdio.h>
3 int num = 0;
4 %}
5 %%
6 \<[a-zA-Z0-9]+>" printf("%s is valid markup tag \n",yytext);
7 "<!--(.|\n)*-->" num++;
8 \n ;
9 . ;
10 %%
11 int main(){
12 yyin=fopen("input.txt","r");
13 yylex();
14 printf("%d comment", num);
15 return 0;
16 }
17 int yywrap(){ return(1);}
```

**Output :**

Name	Date Modified	Size	Kind
a.out	Today at 3:51PM	53 KB	Document
d5.l	Today at 3:50PM	259 bytes	text document
input.txt	Today at 3:49PM	74 bytes	Plain Text
lex.yy.c	Today at 3:51PM	45 KB	text document

```
[ompandit@Oms-MacBook-Pro ~ % cd Desktop
[ompandit@Oms-MacBook-Pro Desktop % cd SEM\ 6
[ompandit@Oms-MacBook-Pro SEM 6 % cd COMPILER\ DESIGN
[ompandit@Oms-MacBook-Pro COMPILER DESIGN % cd P4-5
[ompandit@Oms-MacBook-Pro P4-5 % flex d5.1
[ompandit@Oms-MacBook-Pro P4-5 % gcc lex.yy.c
[ompandit@Oms-MacBook-Pro P4-5 % ./a.out
<html> is valid markup tag
<head> is valid markup tag
<body> is valid markup tag
ompandit@Oms-MacBook-Pro P4-5 %
```

## Experiment - 5

**Aim :** Perform the following :

- A. Write a Lex program to count the number of C comment lines from a given C program. Also eliminate them and copy that program into separate file.

**Code :**

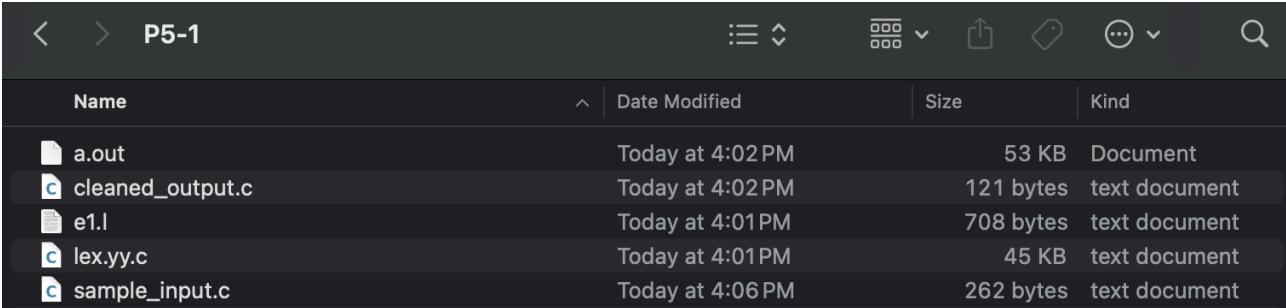
```
P5-1 > C sample_input.c > ...
1  #include <stdio.h>
2
3  // This is a single-line comment
4  int main() {
5      int a = 5; /* Inline multi-line
6      | | | | comment */
7      int b = 10; // Another comment
8      printf("Sum = %d\n", a + b);
9      /* Entire
10         block
11         comment */
12     return 0;
13 }
```

```

P5-1 > e1.l
 1  %{
 2  #include <stdio.h>
 3  int comment_count = 0;
 4  FILE *out;
 5  %}
 6
 7  %%
 8  /*.*                         { comment_count++; /* skip single-line comment */ }
 9  /*(*[^*]|\*+[^\*/])* \*/     { comment_count++; /* skip multi-line comment */ }
10  .|\n                         { fputc(yytext[0], out); }
11  %%
12
13  int main() {
14      FILE *in = fopen("sample_input.c", "r");
15      if (!in) {
16          perror("Input file error");
17          return 1;
18      }
19      out = fopen("cleaned_output.c", "w");
20      if (!out) {
21          perror("Output file error");
22          return 1;
23      }
24      yyin = in;
25      yylex();
26      fclose(in);
27      fclose(out);
28      printf("Total comments removed: %d\n", comment_count);
29      return 0;
30  }
31
32  int yywrap() {
33      return 1;
34  }

```

## Output :



Name	Date Modified	Size	Kind
a.out	Today at 4:02PM	53 KB	Document
cleaned_output.c	Today at 4:02PM	121 bytes	text document
e1.l	Today at 4:01PM	708 bytes	text document
lex.yy.c	Today at 4:01PM	45 KB	text document
sample_input.c	Today at 4:06PM	262 bytes	text document

```
[ompandit@Oms-MacBook-Pro ~ % cd Desktop
[ompandit@Oms-MacBook-Pro Desktop % cd SEM\ 6
[ompandit@Oms-MacBook-Pro SEM 6 % cd COMPILER\ DESIGN
[ompandit@Oms-MacBook-Pro COMPILER DESIGN % cd P5-1
[ompandit@Oms-MacBook-Pro P5-1 % flex e1.l
[ompandit@Oms-MacBook-Pro P5-1 % gcc lex.yy.c
[ompandit@Oms-MacBook-Pro P5-1 % ./a.out
[ompandit@Oms-MacBook-Pro P5-1 % flex e1.l
[ompandit@Oms-MacBook-Pro P5-1 % gcc lex.yy.c
[ompandit@Oms-MacBook-Pro P5-1 % ./a.out
[ompandit@Oms-MacBook-Pro P5-1 % flex e1.l
[ompandit@Oms-MacBook-Pro P5-1 % gcc lex.yy.c
[ompandit@Oms-MacBook-Pro P5-1 % ./a.out
Total comments removed: 4
ompandit@Oms-MacBook-Pro P5-1 %
```

P5-1 > C cleaned\_output.c > ...

```
1  #include <stdio.h>
2
3
4  int main() {
5      int a = 5;
6      int b = 10;
7      printf("Sum = %d\n", a + b);
8
9      return 0;
10 }
```

B. Write a Lex program to recognise keywords, identifiers, operators, numbers, special symbols, literals from a given C program.

Code :

```
P5-2 > C sample_input.c > ...
1  #include <stdio.h>
2
3  int main() {
4      int a = 10;
5      float b = 3.14;
6      char c = 'x';
7      printf("Hello, World!");
8      if (a > b) {
9          a++;
10     }
11     return 0;
12 }
```

```

%{
#include <stdio.h>
%}

digit      [0-9]
alpha      [a-zA-Z]
id        {alpha}({alpha}|{digit})*
int_const  {digit}+
float_const {digit}+.{digit}+
string_lit  \"([^\\"\\]|\\.)*\\"
char_lit   '\([^\\"\\]|\\.)\'

%%%
"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"  { printf("Keyword: %s\n", yytext); }

{id}        { printf("Identifier: %s\n", yytext); }
"=="|"!="|"<="|">="|"&&"|"||"|"++"|"--"|"+"|"-"|"*"|"/"|"="|"<"|">"|""
{ printf("Operator: %s\n", yytext); }

{float_const}  { printf("Float Number: %s\n", yytext); }
{int_const}    { printf("Integer Number: %s\n", yytext); }
{string_lit}   { printf("String Literal: %s\n", yytext); }
{char_lit}     { printf("Character Literal: %s\n", yytext); }
[{}();,]       { printf("Special Symbol: %s\n", yytext); }
[ \t\n]+      { /* skip whitespace */ }
.             { printf("Unrecognized token: %s\n", yytext); }

%%%
int main() {
    yyin = fopen("sample_input.c", "r");
    if (!yyin) {
        perror("Could not open file");
        return 1;
    }
    yylex();
    fclose(yyin);
    return 0;
}

int yywrap() {
    return 1;
}

```

**Output :**

Name	Date Modified	Size	Kind
a.out	Today at 4:30 PM	53 KB	Document
e2.l	Today at 4:30 PM	1 KB	text document
lex.yy.c	Today at 4:30 PM	52 KB	text document
sample_input.c	Today at 4:31PM	170 bytes	text document

```
[ompandit@Oms-MacBook-Pro ~ % cd Desktop
[ompandit@Oms-MacBook-Pro Desktop % cd SEM\ 6
[ompandit@Oms-MacBook-Pro SEM 6 % cd COMPILER\ DESIGN
[ompandit@Oms-MacBook-Pro COMPILER DESIGN % cd P5-2
[ompandit@Oms-MacBook-Pro P5-2 % flex e2.l
[ompandit@Oms-MacBook-Pro P5-2 % gcc lex.yy.c
[ompandit@Oms-MacBook-Pro P5-2 % ./a.out
Unrecognized token: #
Identifier: include
Operator: <
Identifier: stdio
Unrecognized token: .
Identifier: h
Operator: >
Keyword: int
Identifier: main
Special Symbol: (
Special Symbol: )
Special Symbol: {
Keyword: int
Identifier: a
Operator: =
Integer Number: 10
Special Symbol: ;
Keyword: float
Identifier: b
Operator: =
Float Number: 3.14
Special Symbol: ;
Keyword: char
Identifier: c
Operator: =
Character Literal: 'x'
Special Symbol: ;
Identifier: printf
Special Symbol: (
String Literal: "Hello, World!"
Special Symbol: )
Special Symbol: ;
Keyword: if
Special Symbol: (
Identifier: a
Operator: >
Identifier: b
Special Symbol: )
Special Symbol: {
Identifier: a
Operator: ++
Special Symbol: ;
Special Symbol: }
Keyword: return
Integer Number: 0
Special Symbol: ;
Special Symbol: }
ompandit@Oms-MacBook-Pro P5-2 %
```

## Experiment - 6

**Aim :** Program to implement Recursive Descent Parsing in C.

**Code :**

```
#include<stdio.h>
#include<stdlib.h>
/*
E-> iE
E-> +iE / -iE / epsilon
*/
char s[20];
int i=1;
char l;
int match(char t)
{
    if(l==t){
        l=s[i];
        i++;
    }
    else{
        printf("Syntax error");
        exit(1);
    }
}

int E_()
{
    if(l=='+' ){
        match('+');
        match('i');
        E_();
    }
    else if(l=='-' ){
        match('-');
        match('i');
        E_();
    }
    else
        return(1);
}

int E()
{
    if(l=='i'){
        match('i');
        E_();
    }
}

int main()
{
    printf("\nEnter the set of characters to be checked :");
    scanf("%s",&s);
    l=s[0];
    E();
    if(l=='$')
}
```

```
{  
    printf("Success \n");  
}  
else{  
    printf("syntax error");  
}  
return 0;  
}
```

## Output :

```
/Users/ompandit/Desktop/SEM\ 6/COMPILER\ DESIGN/YACC/rd ; exit;  
ompandit@Oms-MacBook-Pro ~ % /Users/ompandit/Desktop/SEM\ 6/COMPILER\ DESIGN/YACC/rd ; exit;  
  
Enter the set of characters to be checked :i+i$  
Success  
  
Saving session...  
...copying shared history...  
...saving history...truncating history files...  
...completed.  
Deleting expired sessions... 3 completed.  
  
[Process completed]
```

## Experiment - 7

**Aim :** Perform the following :

- A. Create Yacc and Lex specification files to recognises arithmetic expressions involving +, -, \* and / .

**Code :**

```
1 >  ≡ lex.l
1   %{
2   #include "yacc.tab.h"
3   #include <stdlib.h>
4   #include <string.h>
5   %}
6
7   %%
8   [0-9]+           { yyval = atoi(yytext); return NUMBER; }
9   [a-zA-Z_][a-zA-Z0-9_]* { return IDENTIFIER; }
10  [+\\-*/]          { return yytext[0]; }
11  \\n              { return '\\n'; }
12  [ \\t]            { /* skip whitespace */ }
13  .                { return 0; } // Invalid character triggers error in yacc
14  %%
15
16  int yywrap() { return 1; }
```

```
1 > ≡ yacc.y
 1   %{
 2   #include <stdio.h>
 3   #include <stdlib.h>
 4
 5   int yylex(void);
 6   int yyerror(char *s);
 7   %}
 8
 9   %token NUMBER IDENTIFIER
10
11  %left '+' '-'
12  %left '*' '/'
13
14  %%
15  stmt: expr '\n' { printf("Valid\n"); }
16  |
17
18  expr: expr '+' expr
19  |
20  | expr '-' expr
21  |
22  | expr '*' expr
23  |
24  | expr '/' expr
25  |
26  | NUMBER
27  |
28  | IDENTIFIER
29  |
30
31  int yyerror(char *s) {
32  |
33  | printf("Invalid\n");
34  | return 0;
35 }
```

**Output :**

```
[ompandit@Oms-MacBook-Pro ~ % cd Desktop
[ompandit@Oms-MacBook-Pro Desktop % cd SEM\ 6
[ompandit@Oms-MacBook-Pro SEM 6 % cd COMPILER\ DESIGN
[ompandit@Oms-MacBook-Pro COMPILER DESIGN % cd YACC
[ompandit@Oms-MacBook-Pro YACC % cd 1
[ompandit@Oms-MacBook-Pro 1 % flex lex.l
[ompandit@Oms-MacBook-Pro 1 % bison -d yacc.y
[ompandit@Oms-MacBook-Pro 1 % gcc lex.yy.c yacc.tab.c
[ompandit@Oms-MacBook-Pro 1 % ./a.out
a+b
Valid
```

B. Create Yacc and Lex specification files are used to generate a calculator which accepts integer type arguments.

**Code :**

```
2 > ≡ lex.l
1   %{
2   #include "yacc.tab.h"
3   %}
4
5   %%
6   [0-9]+     { yyval = atoi(yytext); return NUMBER; }
7   [+\\-*\\n]   { return yytext[0]; }
8   [ \\t]      { /* ignore whitespace */ }
9   %%
10
11  int yywrap() { return 1; }
```

```
2 > ≡ yacc.y
 1  %{
 2  #include <stdio.h>
 3  #include <stdlib.h>
 4
 5  int yylex(void);
 6  int yyerror(char *s);
 7  %}
 8
 9  %token NUMBER
10
11 // Precedence and associativity rules
12 %left '+' '-'
13 %left '*' '/'
14
15 %%
16 stmt: expr '\n' { printf("Result = %d\n", $1); }
17 | ;
18
19 expr: expr '+' expr { $$ = $1 + $3; }
20 | expr '-' expr { $$ = $1 - $3; }
21 | expr '*' expr { $$ = $1 * $3; }
22 | expr '/' expr {
23     if ($3 == 0) {
24         printf("Error: Divide by zero\n");
25         exit(1);
26     }
27     $$ = $1 / $3;
28 }
29 | NUMBER
30 ;
31 %%
32
33 int main() {
34     return yyparse();
35 }
36
37 int yyerror(char *s) {
38     printf("Error: %s\n", s);
39     return 0;
40 }
```

**Output :**

```
[ompandit@Oms-MacBook-Pro ~ % cd Desktop
[ompandit@Oms-MacBook-Pro Desktop % cd SEM\ 6
[ompandit@Oms-MacBook-Pro SEM 6 % cd COMPILER\ DESIGN
[ompandit@Oms-MacBook-Pro COMPILER DESIGN % cd YACC
[ompandit@Oms-MacBook-Pro YACC % cd 2
[ompandit@Oms-MacBook-Pro 2 % flex lex.l
[ompandit@Oms-MacBook-Pro 2 % bison -d yacc.y
[ompandit@Oms-MacBook-Pro 2 % gcc lex.yy.c yacc.tab.c
[ompandit@Oms-MacBook-Pro 2 % ./a.out
3+4*2-1
Result = 10
```

C. Create Yacc and Lex specification files are used to convert infix expression to postfix expression.

**Code :**

```
3 > ≡ lex.l
1   %{
2   #include "yacc.tab.h"
3   #include <stdlib.h>
4   #include <string.h>
5   %}
6
7   %%
8   [0-9]+           { yyval.num = atoi(yytext); return NUMBER; }
9   [a-zA-Z_][a-zA-Z0-9_]* { yyval.id = strdup(yytext); return IDENTIFIER; }
10  [+\\-*/()\\n]    { return yytext[0]; }
11  [ \\t]+          { /* ignore whitespace */ }
12  .                { printf("Invalid character: %s\\n", yytext); return -1; }
13  %%
14
15  int yywrap() { return 1; }
```

```
3 >  ≡ yacc.y
 1   %{
 2   #include <stdio.h>
 3   #include <stdlib.h>
 4
 5   int yylex(void);
 6   int yyerror(char *s);
 7   %}
 8
 9   %union {
10   |   int num;
11   |   char* id;
12   }
13
14   %token <num> NUMBER
15   %token <id> IDENTIFIER
16
17   %%
18   stmt: expr '\n' { printf("\n"); }
19   |
20
21   expr: expr '+' term   { printf("+ ");}
22   | expr '-' term   { printf("- ");}
23   | term
24   |
25
26   term: term '*' factor { printf("* ");}
27   | term '/' factor { printf("/ ");}
28   | factor
29   |
30
31   factor: '(' expr ')'
32   | NUMBER          { printf("%d ", $1); }
33   | IDENTIFIER      { printf("%s ", $1); free($1); }
34   |
35   %%
36
37   int main() {
38   |   return yyparse();
39   }
40
41   int yyerror(char *s) {
42   |   fprintf("Error: %s\n", s);
43   |   return 0;
44   }
```

**Output :**

```
[ompandit@Oms-MacBook-Pro ~ % cd Desktop
[ompandit@Oms-MacBook-Pro Desktop % cd SEM\ 6
[ompandit@Oms-MacBook-Pro SEM 6 % cd COMPILER\ DESIGN
[ompandit@Oms-MacBook-Pro COMPILER DESIGN % cd YACC
[ompandit@Oms-MacBook-Pro YACC % cd 3
[ompandit@Oms-MacBook-Pro 3 % flex lex.l
[ompandit@Oms-MacBook-Pro 3 % bison -d yacc.y
[ompandit@Oms-MacBook-Pro 3 % gcc lex.yy.c yacc.tab.c
[ompandit@Oms-MacBook-Pro 3 % ./a.out
a+5-b
a 5 + b -
[
```