



## **COMPILER DESIGN LAB FILE**

Submitted by: Anshi

SAP ID:500101953

Batch: B4

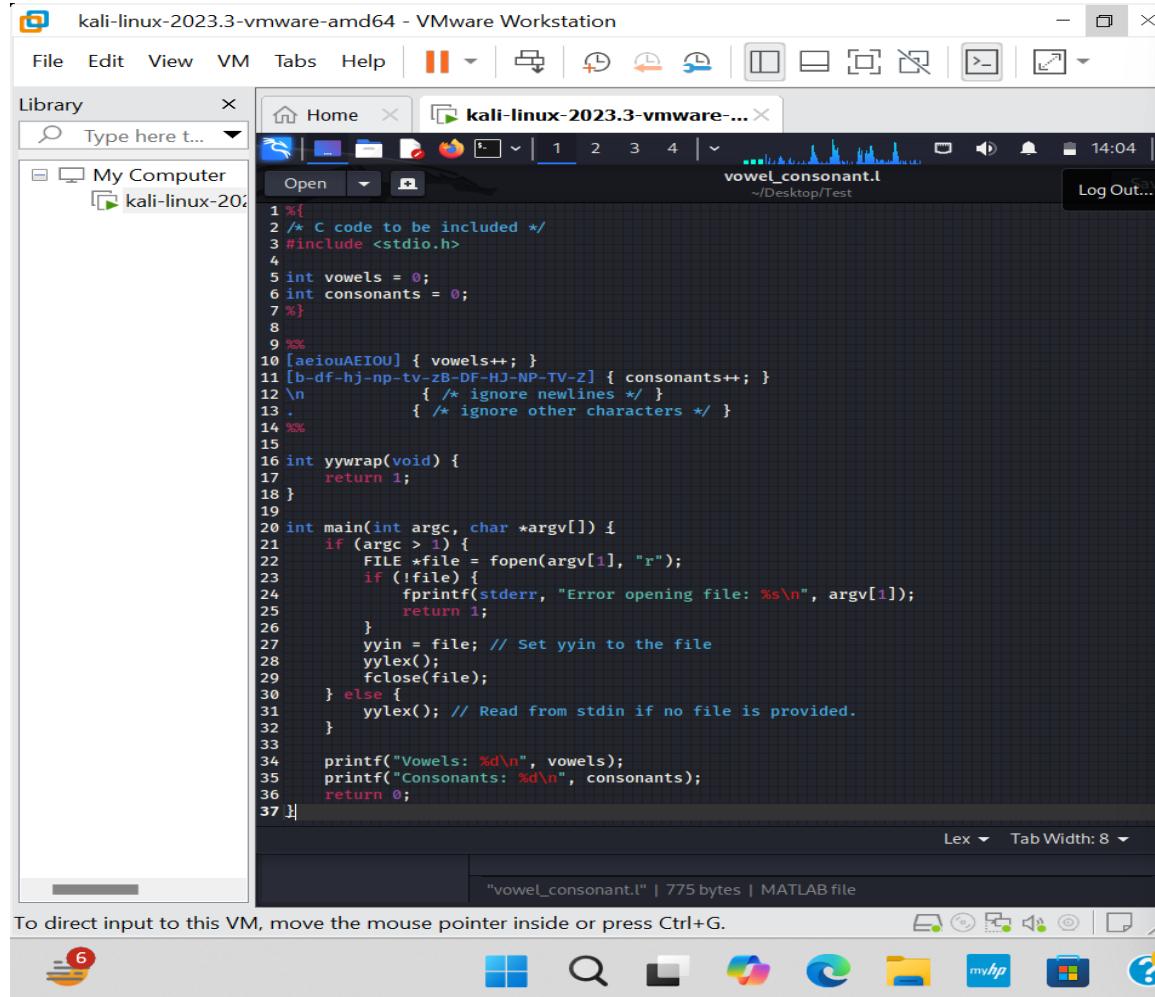
## INDEX

<u>Experiment no:</u>	<u>Title</u>	<u>Page no:</u>	<u>Remarks</u>
2	<b>Lex Tools</b>	<b>3-12</b>	
3	<b>Lexical Analysis</b>	<b>13-19</b>	
4	<b>Syntax Analysis</b>	<b>20-26</b>	
5	<b>First and Follow</b>	<b>27-29</b>	
6	<b>LL(1) Parser</b>	<b>30-31</b>	
7	<b>Predictive Parser</b>	<b>32-34</b>	
8	<b>Leading</b>	<b>35-37</b>	
9	<b>Trailing</b>	<b>37-39</b>	
10	<b>Operator Precedence</b>	<b>39-41</b>	
11	<b>Shift-reduce parser</b>	<b>41-43</b>	
12	<b>LALR Parser</b>	<b>44-48</b>	

## **EXPERIMENT NO-2**

Title: Introductory Problems using Lex Tools

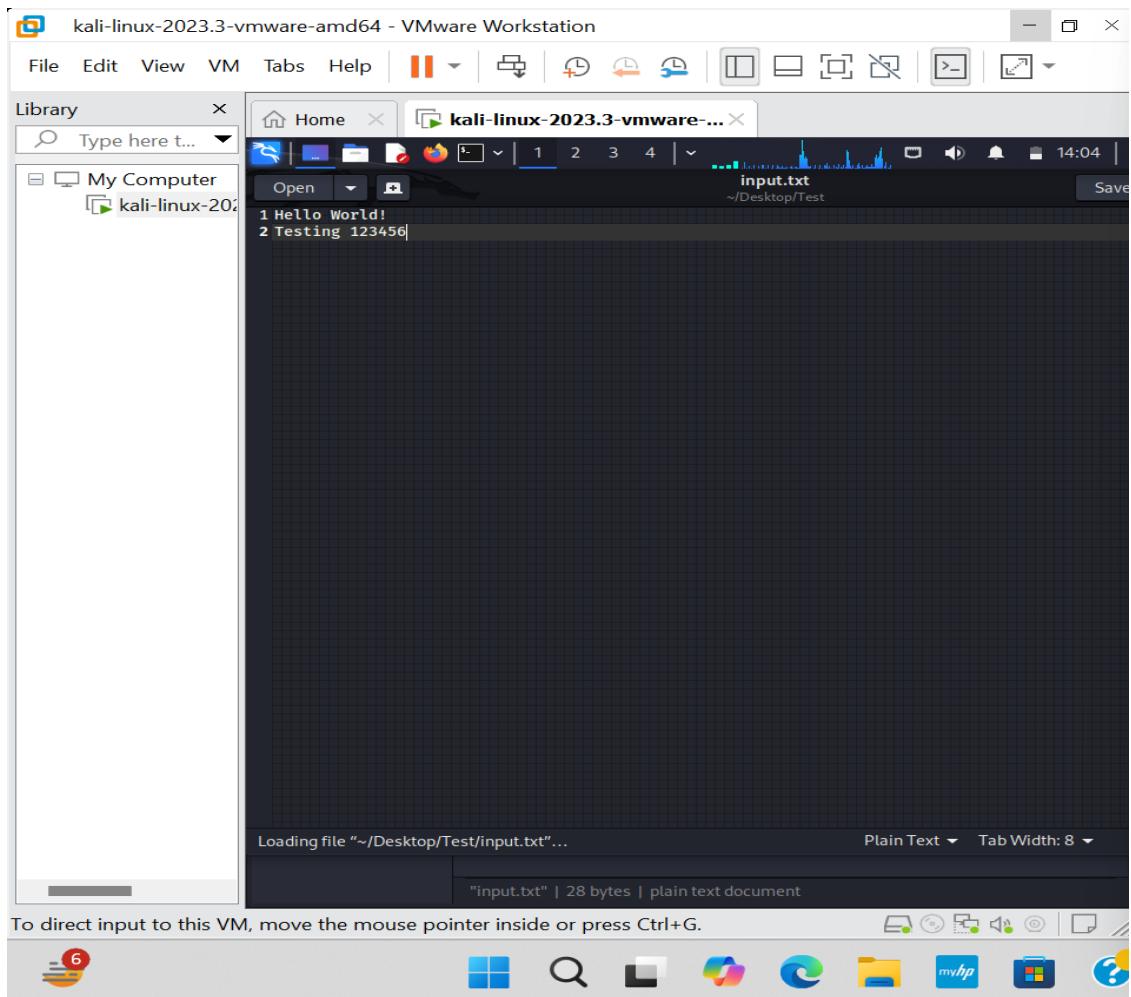
1. WAP to count number of vowels and consonants in a given string.

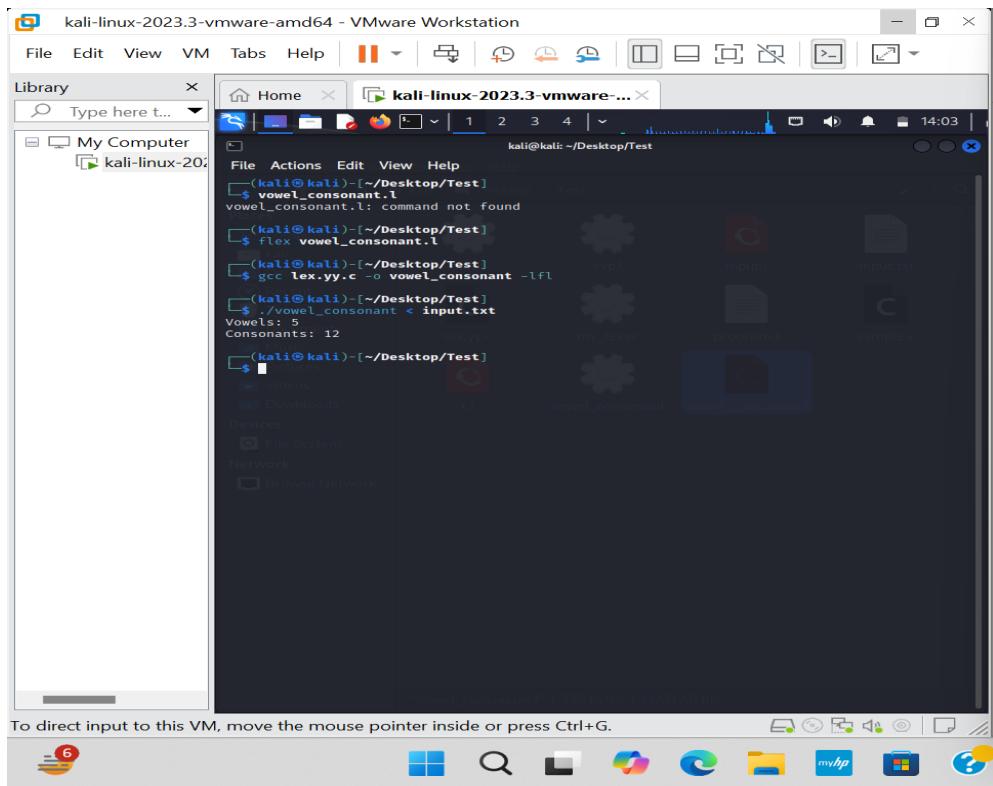


```
1 %}
2 /* C code to be included */
3 #include <stdio.h>
4
5 int vowels = 0;
6 int consonants = 0;
7 %%
8
9 %%
10 [aeiouAEIOU] { vowels++; }
11 [b-df-hj-np-tv-zA-Z] { consonants++; }
12 \n          { /* ignore newlines */ }
13 .           { /* ignore other characters */ }
14 %%
15
16 int yywrap(void) {
17     return 1;
18 }
19
20 int main(int argc, char *argv[]) {
21     if (argc > 1) {
22         FILE *file = fopen(argv[1], "r");
23         if (!file) {
24             fprintf(stderr, "Error opening file: %s\n", argv[1]);
25             return 1;
26         }
27         yyin = file; // Set yyin to the file
28         yylex();
29         fclose(file);
30     } else {
31         yylex(); // Read from stdin if no file is provided.
32     }
33
34     printf("Vowels: %d\n", vowels);
35     printf("Consonants: %d\n", consonants);
36     return 0;
37 }
```

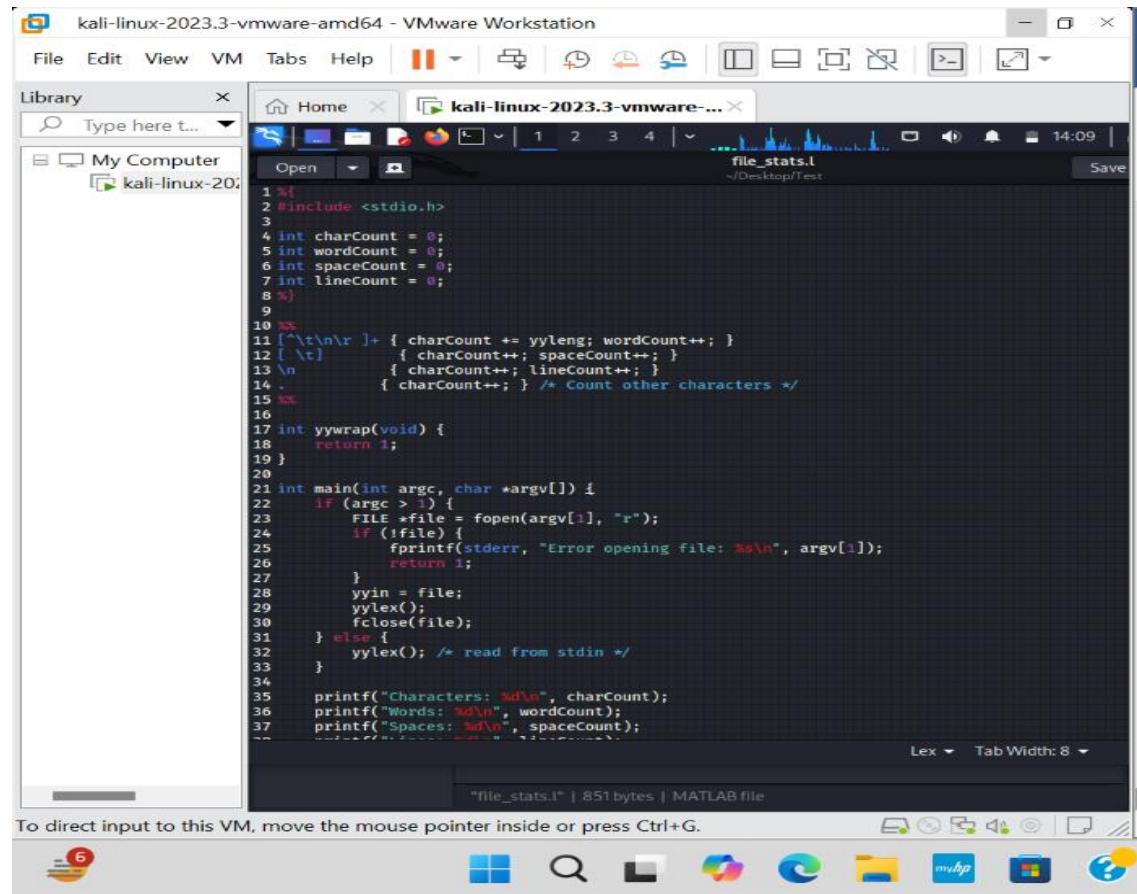
To direct input to this VM, move the mouse pointer inside or press Ctrl+G.







2. WAP to count the number of characters, words, spaces, and end of lines in a given input file.



```
1 %{
2 #include <stdio.h>
3
4 int charCount = 0;
5 int wordCount = 0;
6 int spaceCount = 0;
7 int lineCount = 0;
8 %
9
10 /**
11 [^\t\n\r ]+ { charCount += yylen; wordCount++; }
12 [\t] { charCount++; spaceCount++; }
13 \n { charCount++; lineCount++; }
14 . { charCount++; } /* count other characters */
15 */
16
17 int yywrap(void) {
18     return 1;
19 }
20
21 int main(int argc, char *argv[]) {
22     if (argc > 1) {
23         FILE *file = fopen(argv[1], "r");
24         if (!file) {
25             fprintf(stderr, "Error opening file: %s\n", argv[1]);
26             return 1;
27         }
28         yyin = file;
29         yylex();
30         fclose(file);
31     } else {
32         yylex(); /* read from stdin */
33     }
34
35     printf("Characters: %d\n", charCount);
36     printf("Words: %d\n", wordCount);
37     printf("Spaces: %d\n", spaceCount);
38 }
```

To direct input to this VM, move the mouse pointer inside or press Ctrl+G.

kali-linux-2023.3-vmware-amd64 - VMware Workstation

File Edit View VM Tabs Help | 1 2 3 4 | Open Save | 14:09 |

Library Type here t... My Computer kali-linux-20...

file\_stats.l ~/Desktop/Test

Save

```
5 int wordCount = 0;
6 int spaceCount = 0;
7 int lineCount = 0;
8 %
9 /**
10 * @param[in] file - The file to read from.
11 * @return The number of words in the file.
12 */
13 int wordCount(FILE *file) {
14     int c, prevSpace = 1, wordStart = 1;
15     while ((c = fgetc(file)) != EOF) {
16         if (c == ' ' || c == '\t' || c == '\n') {
17             if (prevSpace) wordStart = 1;
18             prevSpace = 1;
19         } else {
20             prevSpace = 0;
21         }
22     }
23     return wordStart ? wordStart : 0;
24 }
25
26 int main(int argc, char *argv[]) {
27     if (argc > 1) {
28         FILE *file = fopen(argv[1], "r");
29         if (!file) {
30             fprintf(stderr, "Error opening file: %s\n", argv[1]);
31             exit(1);
32         }
33         yylex();
34         fclose(file);
35     } else {
36         yylex(); /* read from stdin */
37     }
38     printf("Characters: %d\n", charCount);
39     printf("Words: %d\n", wordCount);
40     printf("Spaces: %d\n", spaceCount);
41     printf("Lines: %d\n", lineCount);
42     return 0;
43 }
```

Lex Tab Width: 8

"file\_stats.l" | 851 bytes | MATLAB file

To direct input to this VM, move the mouse pointer inside or press Ctrl+G.

Windows Start button Taskbar icons

kali-linux-2023.3-vmware-amd64 - VMware Workstation

File Edit View VM Tabs Help | 1 2 3 4 | Open Save | 14:08 |

Library Type here t... My Computer kali-linux-20...

input.txt ~/Desktop/Test

Save

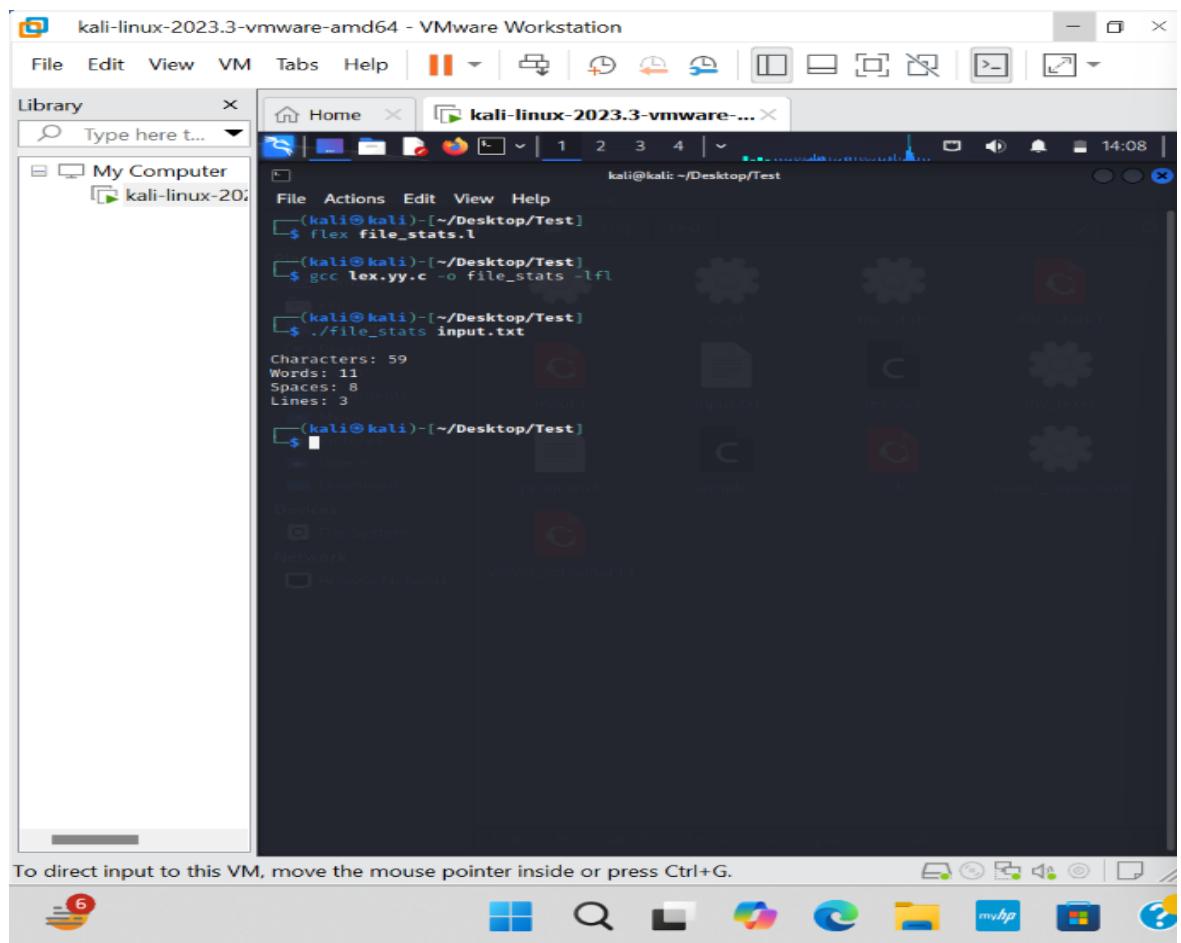
```
1 Hello World!
2 Testing 123456
3 Compiler Design Lab File ! # %
```

Plain Text Tab Width: 8

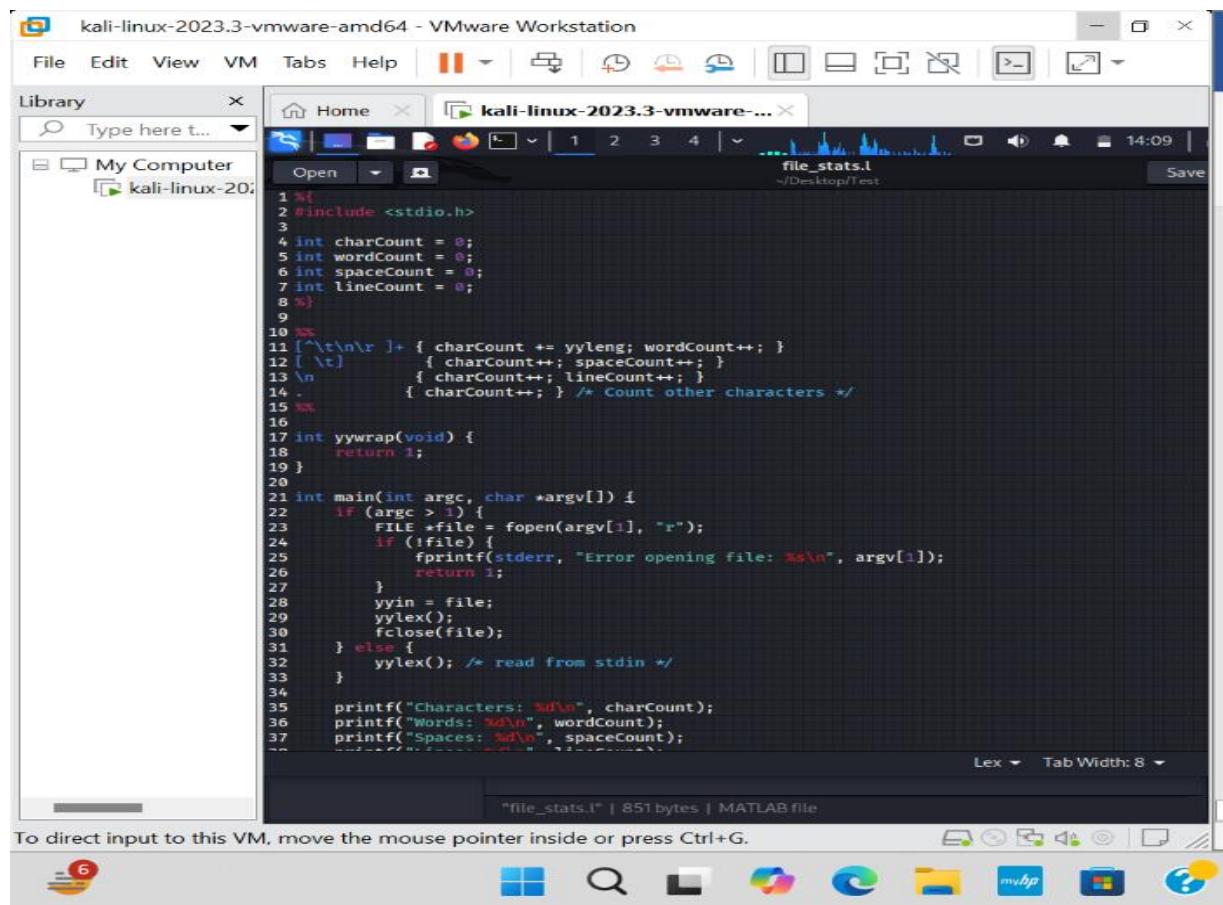
"input.txt" | 59 bytes | plain text document

To direct input to this VM, move the mouse pointer inside or press Ctrl+G.

Windows Start button Taskbar icons



### 3. WAP to count number of comment lines in a given C program.



The screenshot shows a VMware Workstation window titled "kali-linux-2023.3-vmware-amd64 - VMware Workstation". Inside the window, there is a terminal-like interface with a file browser window open. The file browser shows a folder structure with "My Computer" and "kali-linux-20". A file named "file\_stats.l" is selected in the terminal window. The terminal content is a C program:

```
1 /*
2 #include <stdio.h>
3
4 int charCount = 0;
5 int wordCount = 0;
6 int spaceCount = 0;
7 int lineCount = 0;
8 */
9
10 /**
11 [^\t\n\r ]+ { charCount += yylen; wordCount++; }
12 [ \t] { charCount++; spaceCount++; }
13 [\n] { charCount++; lineCount++; }
14 . { charCount++; } /* Count other characters */
15 */
16
17 int yywrap(void) {
18     return 1;
19 }
20
21 int main(int argc, char *argv[]) {
22     if (argc > 1) {
23         FILE *file = fopen(argv[1], "r");
24         if (!file) {
25             fprintf(stderr, "Error opening file: %s\n", argv[1]);
26             return 1;
27         }
28         yyin = file;
29         yylex();
30         fclose(file);
31     } else {
32         yylex(); /* read from stdin */
33     }
34
35     printf("Characters: %d\n", charCount);
36     printf("Words: %d\n", wordCount);
37     printf("Spaces: %d\n", spaceCount);

```

The terminal also displays the message "To direct input to this VM, move the mouse pointer inside or press Ctrl+G." at the bottom.

kali-linux-2023.3-vmware-amd64 - VMware Workstation

File Edit View VM Tabs Help

Library > Type here t... My Computer kali-linux-20...

Open file\_stats.l ~/Desktop/Test

Save

```
5 int wordCount = 0;
6 int spaceCount = 0;
7 int lineCount = 0;
8 %
9 /**
10 [^\t\n\r ]+ { charCount += yylen; wordCount++; }
11 [ \t] { charCount++; spaceCount++; }
12 \n { charCount++; lineCount++; }
13 . { charCount++; } /* Count other characters */
14 /**
15 %
16
17 int yywrap(void) {
18     return 1;
19 }
20
21 int main(int argc, char *argv[]) {
22     if (argc > 1) {
23         FILE *file = fopen(argv[1], "r");
24         if (!file) {
25             fprintf(stderr, "Error opening file: %s\n", argv[1]);
26             return 1;
27         }
28         yyin = file;
29         yylex();
30         fclose(file);
31     } else {
32         yylex(); /* read from stdin */
33     }
34
35     printf("Characters: %d\n", charCount);
36     printf("Words: %d\n", wordCount);
37     printf("Spaces: %d\n", spaceCount);
38     printf("Lines: %d\n", lineCount);
39
40     return 0;
41 }
```

Lex Tab Width: 8

"file\_stats.l" | 851 bytes | MATLAB file

To direct input to this VM, move the mouse pointer inside or press Ctrl+G.



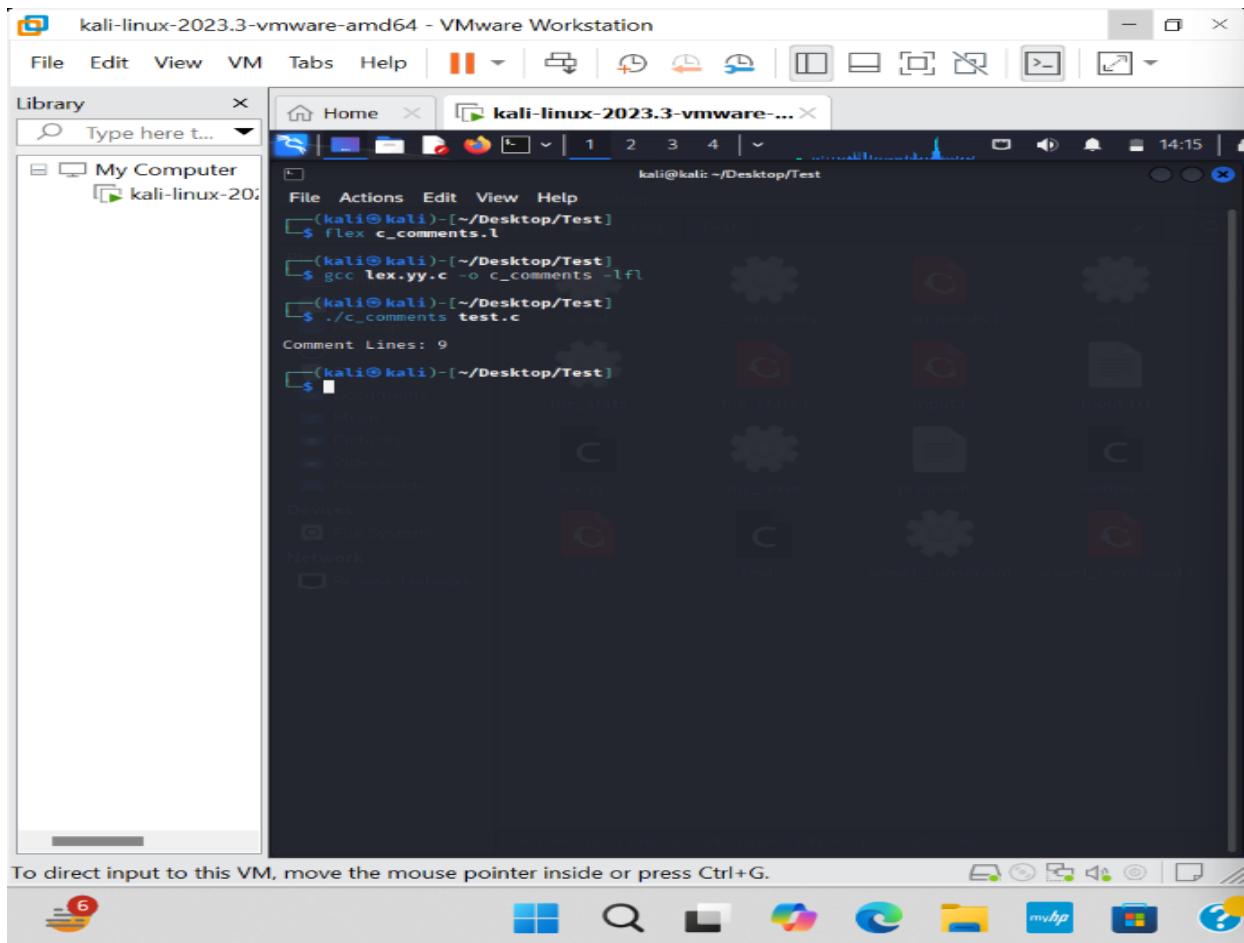
```
File Edit View VM Tabs Help ||| Home kali-linux-2023.3-vmware-... X
Library X
Type here t...
My Computer
kali-linux-2023.3-vmware-...
Open
c_comments.l
Save
1 %{
2 #include <stdio.h>
3
4 int commentLines = 0;
5 int inMultiLineComment = 0;
6 %}
7
8 /**
9 */*.* { commentLines++; /* Corrected single-line comment rule */
10 *//*
11 *//*
12 *\n        { if (inMultiLineComment) commentLines++; }
13 .        /* Ignore other characters */
14 */
15
16 int yywrap(void) {
17     return 1;
18 }
19
20 int main(int argc, char *argv[]) {
21     if (argc > 1) {
22         FILE *file = fopen(argv[1], "r");
23         if (!file) {
24             fprintf(stderr, "Error opening file: %s\n", argv[1]);
25             return 1;
26         }
27         yyin = file;
28         yylex();
29         fclose(file);
30     } else {
31         yylex(); /* read from stdin */
32     }
33
34     printf("Comment Lines: %d\n", commentLines);
35     return 0;
36 }
```

Bracket match found on line: 20

"c\_comments.l" | 781 bytes | MATLAB file

Lex Tab Width: 8

To direct input to this VM, move the mouse pointer inside or press Ctrl+G.



## EXPERIMENT NO-3

Title: Lexical Analysis through Lexx Tools.

1. WAP to count number of ‘scanf’ and ‘printf’ statements in a C program. Replace them with ‘read’ and ‘write’ statements respectively.

```
1 %{
2 #include <stdio.h>
3
4 int commentLines = 0;
5 int inMultiLineComment = 0;
6 %}
7
8 /**
9 /**.* { commentLines++; } /* Corrected single-line comment rule */
10 /*/* { inMultiLineComment = 1; }
11 /**/
12 \n { if (inMultiLineComment) commentLines++; }
13 . /* Ignore other characters */
14 */
15
16 int yywrap(void) {
17     return 1;
18 }
19
20 int main(int argc, char *argv[]) {
21     if (argc > 1) {
22         FILE *file = fopen(argv[1], "r");
23         if (!file) {
24             fprintf(stderr, "Error opening file: %s\n", argv[1]);
25             return 1;
26         }
27         yyin = file;
28         yylex();
29         fclose(file);
30     } else {
31         yylex(); /* read from stdin */
32     }
33     printf("Comment Lines: %d\n", commentLines);
34     return 0;
35 }
```

Brace matching found on line: 20

"c\_comments.l" | 781 bytes | MATLAB file

To direct input to this VM, move the mouse pointer inside or press Ctrl+G.

kali-linux-2023.3-vmware-amd64 - VMware Workstation

File Edit View VM Tabs Help

Library Type here t... My Computer kali-linux-20...

Open test1.c

```
1 #include <stdio.h>
2
3 int main() {
4     int a, b;
5     char name[50];
6
7     printf("Enter your name: ");
8     scanf("%s", name);
9
10    printf("Enter two integers: ");
11    scanf("%d %d", &a, &b);
12
13    printf("Your name is: %s\n", name);
14    printf("Sum of %d and %d is: %d\n", a, b, a + b);
15    printf("This is another printf statement.\n");
16
17    scanf("%d", &a);
18
19    return 0;
20
21 }
```

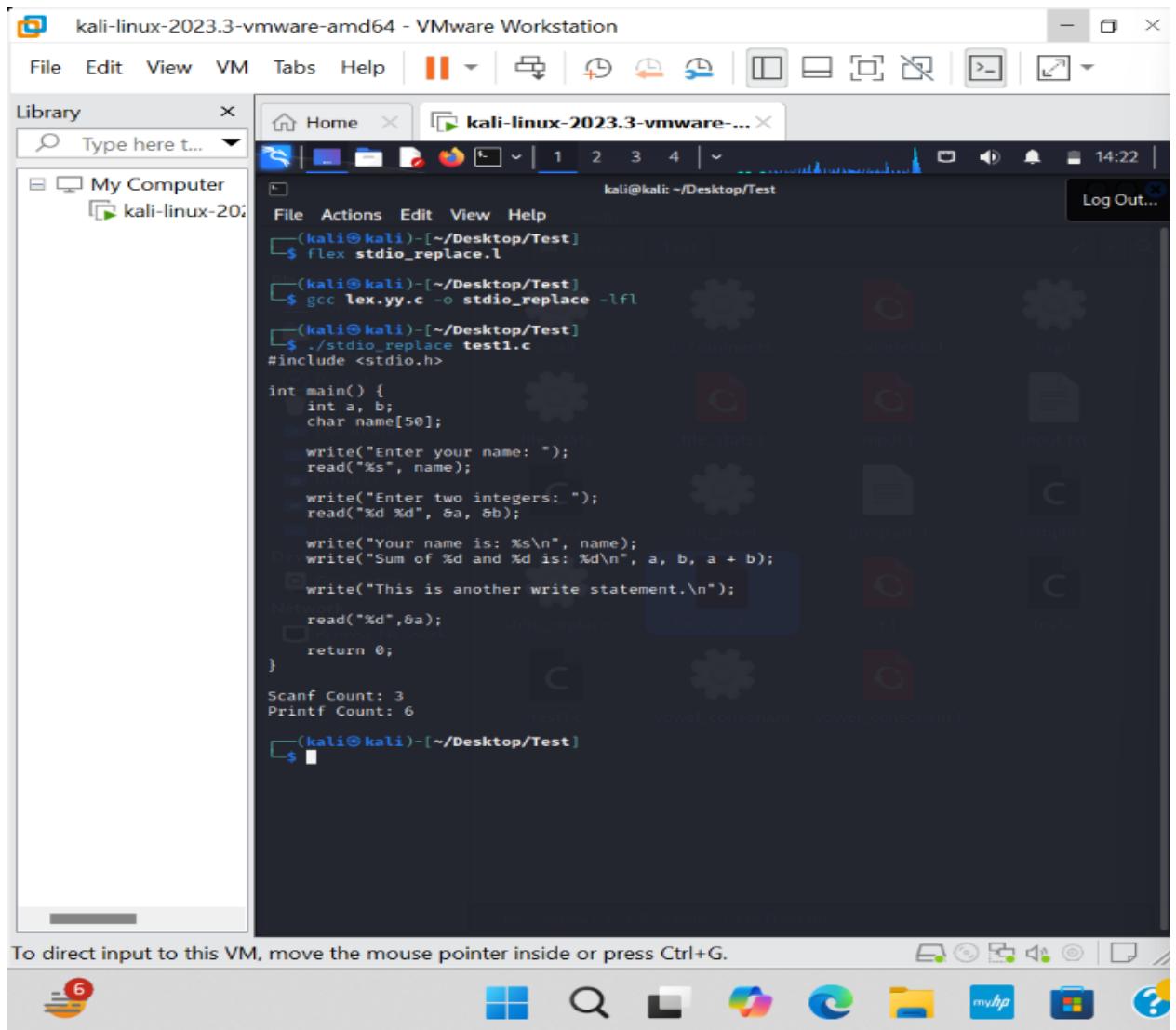
C Tab Width: 8

"test1.c" | 373 bytes | C source code

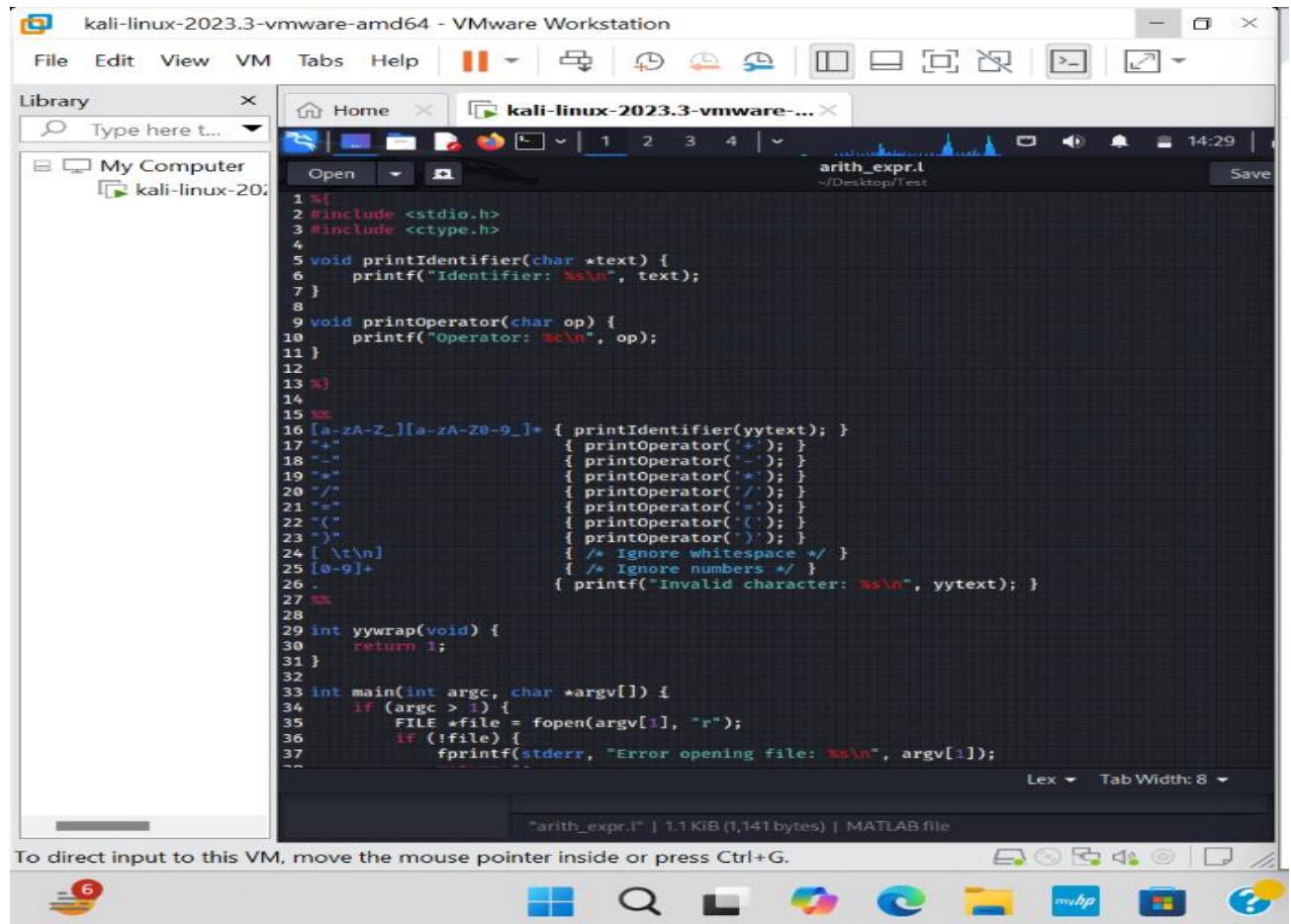
To direct input to this VM, move the mouse pointer inside or press Ctrl+G.



The screenshot shows a VMware Workstation window for a Kali Linux VM. The main area displays a terminal window with a C source code editor showing a simple program named 'test1.c'. The code prompts the user for their name and two integers, then prints them back along with their sum. Below the terminal is a standard Windows-style taskbar with various icons for file operations and system functions.



2. WAP to recognize a valid arithmetic expression and identify the identifiers and operators present. Print them separately.



```
1 /*
2 #include <stdio.h>
3 #include <ctype.h>
4
5 void printIdentifier(char *text) {
6     printf("Identifier: %s\n", text);
7 }
8
9 void printOperator(char op) {
10    printf("Operator: %c\n", op);
11 }
12
13 %
14 %
15 */
16 [a-zA-Z_][a-zA-Z0-9_]* { printIdentifier(yytext); }
17 "+"
18 "-"
19 "*"
20 "/"
21 "="
22 "("
23 ")"
24 [\t\n]
25 [0-9]+
26 .
27 %
28
29 int yywrap(void) {
30     return 1;
31 }
32
33 int main(int argc, char *argv[]) {
34     if (argc > 1) {
35         FILE *file = fopen(argv[1], "r");
36         if (!file) {
37             fprintf(stderr, "Error opening file: %s\n", argv[1]);
38         }
39     }
40 }
```

kali-linux-2023.3-vmware-amd64 - VMware Workstation

File Edit View VM Tabs Help |

Library Type here t...

My Computer

Home 14:29 |

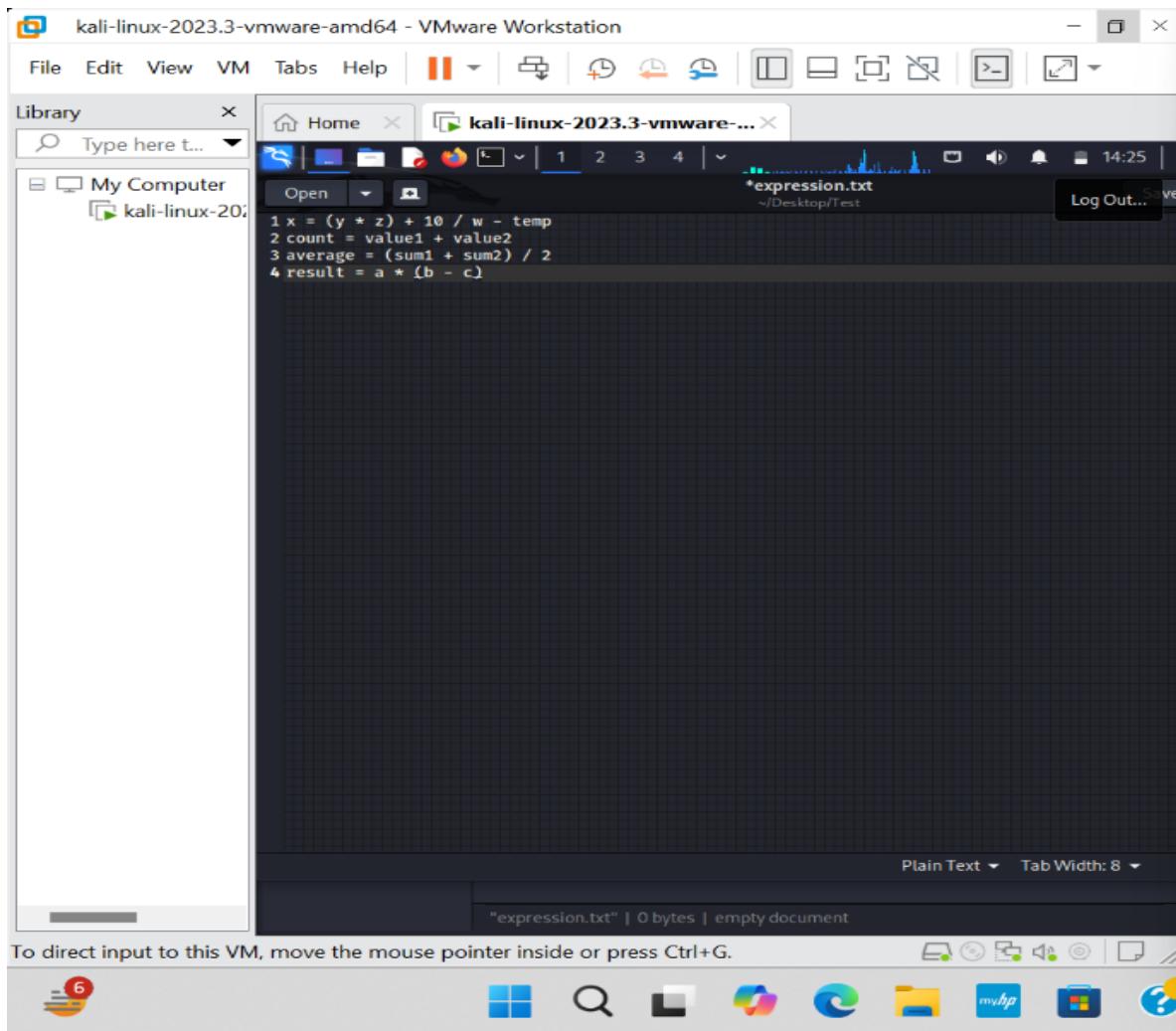
arith\_expr.l ~/Desktop/Test Save

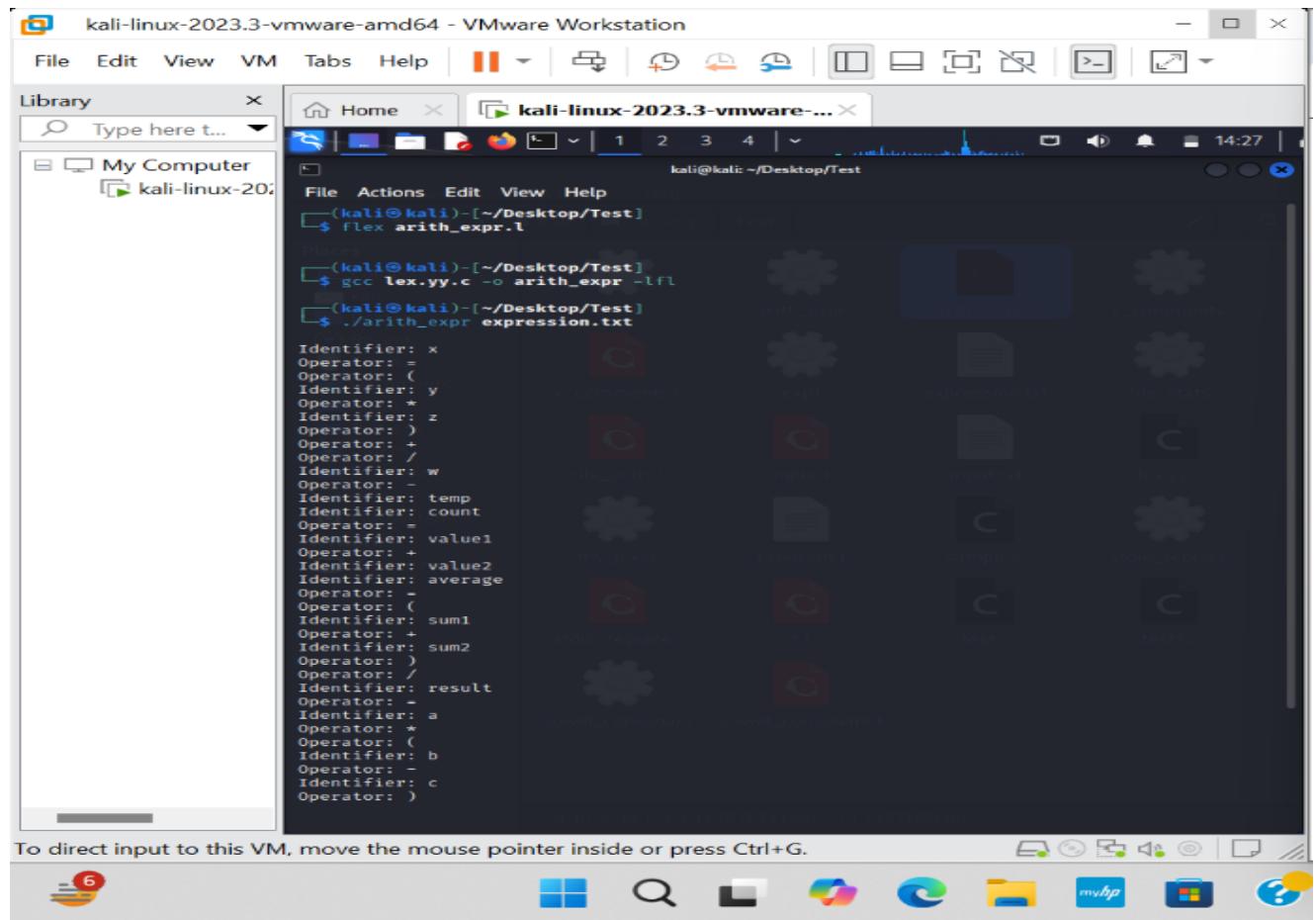
```
11 }
12 }
13 %}
14 }
15 %}
16 [a-zA-Z_][a-zA-Z0-9_]* { printIdentifier(yytext); }
17 "+"
18 "-"
19 "*"
20 "/"
21 "="
22 "("
23 ")"
24 [ \t\n]
25 [0-9]+
26 .
27 %}
28
29 int yywrap(void) {
30     return 1;
31 }
32
33 int main(int argc, char *argv[]) {
34     if (argc > 1) {
35         FILE *file = fopen(argv[1], "r");
36         if (!file) {
37             fprintf(stderr, "Error opening file: %s\n", argv[1]);
38             return 1;
39         }
40         yyin = file;
41         yylex();
42         fclose(file);
43     } else {
44         yylex(); /* read from stdin */
45     }
46     return 0;
47 }
```

Lex Tab Width: 8

"arith\_expr.l" | 1.1 KIB (1,141 bytes) | MATLAB file

To direct input to this VM, move the mouse pointer inside or press Ctrl+G.





## EXPERIMENT NO-4

Title: Syntax Analysis via Yacc Tool.

1. WAP to test the validity of a simple expression involving operators- +, -, \*, and /.

```
1 %{
2 #include <stdio.h>
3 #include <stdlib.h>
4
5 int isValid = 1;
6
7 void invalid(char *msg) {
8     printf("Invalid expression: %s\n", msg);
9     isValid = 0;
10 }
11
12 int checkParentheses(char *str) {
13     int count = 0;
14     for (int i = 0; str[i] != '\0'; i++) {
15         if (str[i] == '(') {
16             count++;
17         } else if (str[i] == ')') {
18             count--;
19         }
20         if (count < 0) {
21             return 0; // Unbalanced parentheses
22         }
23     }
24     return (count == 0); // True if balanced
25 }
26
27 %}
28
29 %option noyywrap
30
31 %%
32
33 [a-zA-Z][a-zA-Z0-9_]*|[0-9]+ { /* Valid identifier or number */ }
34 [+\\-*/()]{ /* Valid operator or parenthesis */ }
35 [ \t\\n]+ { /* Ignore whitespace */ }
36 . { invalid("Invalid character"); }
37 }
```

kali-linux-2023.3-vmware-amd64 - VMware Workstation

File Edit View VM Tabs Help

Library Type here t... My Computer kali-linux-20...

Home kali-linux-2023.3-vmware-... expr\_valid.l ~Desktop/Test

Open Save

```
expr_valid.l
1
26
27 %}
28
29 %option noyywrap
30
31 %% 
32
33 [a-zA-Z_][a-zA-Z0-9_]*[0-9]+ { /* Valid identifier or number */ }
34 [+/*(/)] { /* Valid operator or parenthesis */ }
35 [ \t\n]+ { /* Ignore whitespace */ }
36 . { invalid("Invalid character"); }
37
38 %%
39
40 int main(int argc, char *argv[]) {
41     if (argc > 1) {
42         FILE *file = fopen(argv[1], "r");
43         if (!file) {
44             fprintf(stderr, "Error opening file: %s\n", argv[1]);
45             return 1;
46         }
47         yyin = file;
48         yylex();
49         fclose(file);
50     } else {
51         yylex(); /* read from stdin */
52     }
53
54     if (isValid) {
55         if (checkParentheses(yytext))
56             printf("Valid expression\n");
57         else
58             printf("Invalid expression: Unbalanced parentheses\n");
59     }
60
61     return 0;
62 }
```

Lex Tab Width: 8

"expr\_valid.l" | 1.3 KIB (1,342 bytes) | MATLAB file

To direct input to this VM, move the mouse pointer inside or press Ctrl+G.

Windows Start button Taskbar

kali-linux-2023.3-vmware-amd64 - VMware Workstation

File Edit View VM Tabs Help

Library Type here t... My Computer kali-linux-20...

Home kali-linux-2023.3-vmware-... expression.txt ~Desktop/Test

Open Save Log Out...

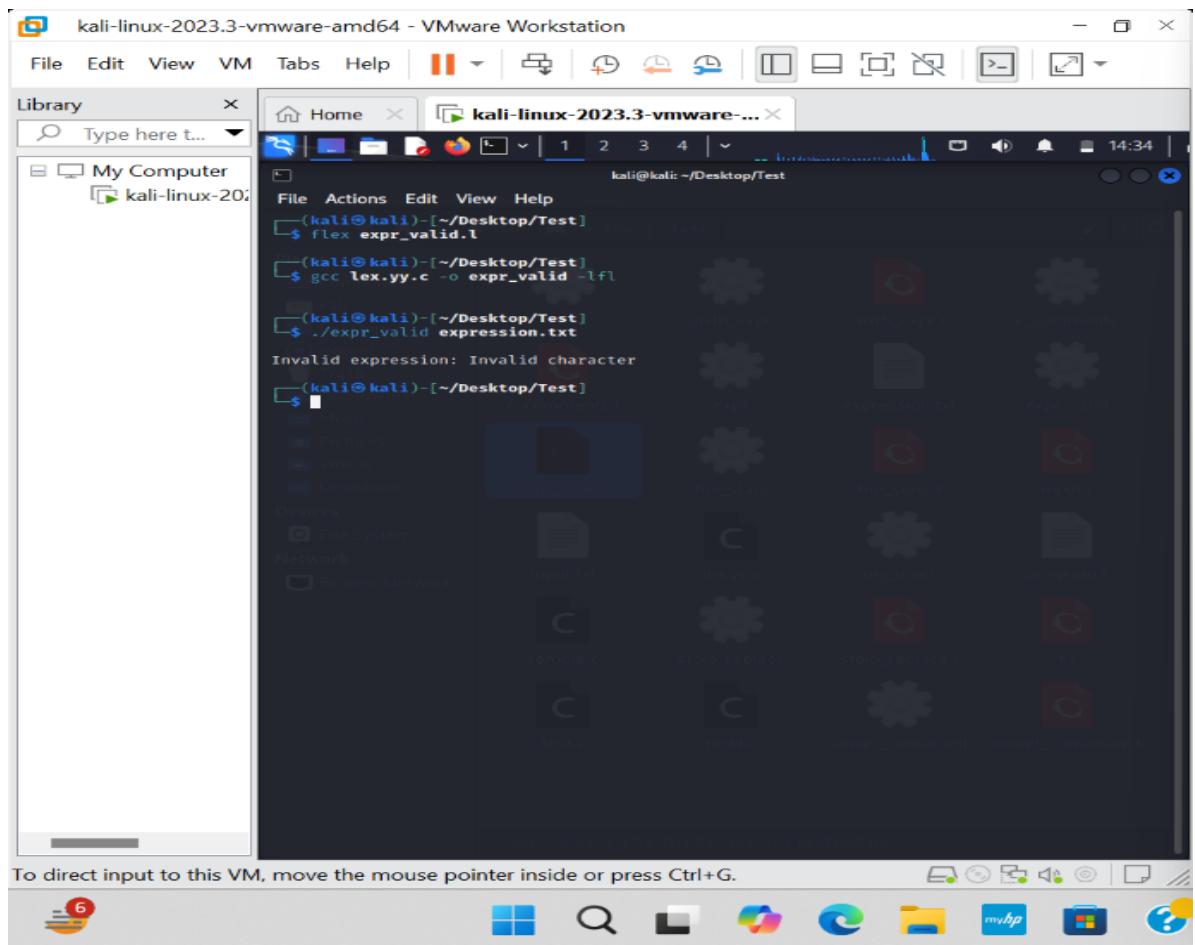
```
expression.txt
1 (a + b) * c / 10 - d
2 10 + 5 * (3 - 2)
3 10 + a * (b - c)
4 10 + a # b
```

Plain Text Tab Width: 8

"expression.txt" | 65 bytes | plain text document

To direct input to this VM, move the mouse pointer inside or press Ctrl+G.

Windows Start button Taskbar



## 2. WAP to evaluate an arithmetic expression involving operators- +, -, \*, and /.

```
1 /*
2  *include <stdio.h>
3  *include <stdlib.h>
4  *include <math.h>
5
6 #define MAX_STACK_SIZE 100
7
8 double stack[MAX_STACK_SIZE];
9 int top = -1;
10
11 void push(double value) {
12     if (top >= MAX_STACK_SIZE - 1) {
13         fprintf(stderr, "Stack overflow\n");
14         exit(1);
15     }
16     stack[++top] = value;
17 }
18
19 double pop() {
20     if (top < 0) {
21         fprintf(stderr, "Stack underflow\n");
22         exit(1);
23     }
24     return stack[top--];
25 }
26
27 double performOperation(char op) {
28     double operand2 = pop();
29     double operand1 = pop();
30     switch (op) {
31         case '+': return operand1 + operand2;
32         case '-': return operand1 - operand2;
33         case '*': return operand1 * operand2;
34         case '/':
35             if (operand2 == 0) {
36                 fprintf(stderr, "Division by zero\n");
37                 exit(1);
38             }
39     }
40 }
```

kali-linux-2023.3-vmware-amd64 - VMware Workstation

File Edit View VM Tabs Help | 14:41

Library Type here to...

My Computer kali-linux-20...

Open expr2\_evalt  
/Desktop/Test

```
27 double performOperation(char op) {
28     double operand2 = pop();
29     double operand1 = pop();
30     switch (op) {
31         case '+': return operand1 + operand2;
32         case '-': return operand1 - operand2;
33         case '*': return operand1 * operand2;
34         case '/':
35             if (operand2 == 0) {
36                 fprintf(stderr, "Division by zero\n");
37                 exit(1);
38             }
39             return operand1 / operand2;
40         default:
41             fprintf(stderr, "Invalid operator\n");
42             exit(1);
43     }
44 }
45
46 %}
47
48 %option noyywrap
49
50 %%
```

51
52 [0-9]+(\.[0-9]+)? { push(atof(yytext)); }
53 [+|-\*/]
54 [ \t\n] { /\* Ignore whitespace \*/ }
55 .
56 { fprintf(stderr, "Invalid token: %s\n", yytext); exit(1); }

57 %%
58
59 int main(int argc, char \*argv[]) {
60 if (argc > 1) {
61 FILE \*file = fopen(argv[1], "r");
62 if (!file) {
63 fprintf(stderr, "Error opening file: %s\n", argv[1]);

Lex Tab Width: 8

unusual whitespace unusual whitespace  
"expr2\_evalt" | 1.7 KB (1,710 bytes) | MATLAB file

To direct input to this VM, move the mouse pointer inside or press Ctrl+G.

A screenshot of a VMware Workstation window titled "kali-linux-2023.3-vmware-amd64 - VMware Workstation". The window contains a terminal and a file browser.

The terminal window shows the following lex code:

```
45
46 %}
47
48 %%option noyywrap
49
50 %%
51
52 [0-9]+(\.[0-9]+)? { push(atof(yytext)); }
53 [\t\n\r\f\v] { /* Ignore whitespace */ }
54 . { fprintf(stderr, "Invalid token: %s\n", yytext); exit(1); }
55
56 int main(int argc, char *argv[]) {
57     if (argc > 1) {
58         FILE *file = fopen(argv[1], "r");
59         if (!file) {
60             fprintf(stderr, "Error opening file: %s\n", argv[1]);
61             return 1;
62         }
63         yyin = file;
64         yylex();
65         fclose(file);
66     } else {
67         yylex(); /* read from stdin */
68     }
69     if (top == e) {
70         printf("Result: %f\n", pop());
71     } else {
72         fprintf(stderr, "Invalid expression\n");
73     }
74     return 0;
75 }
```

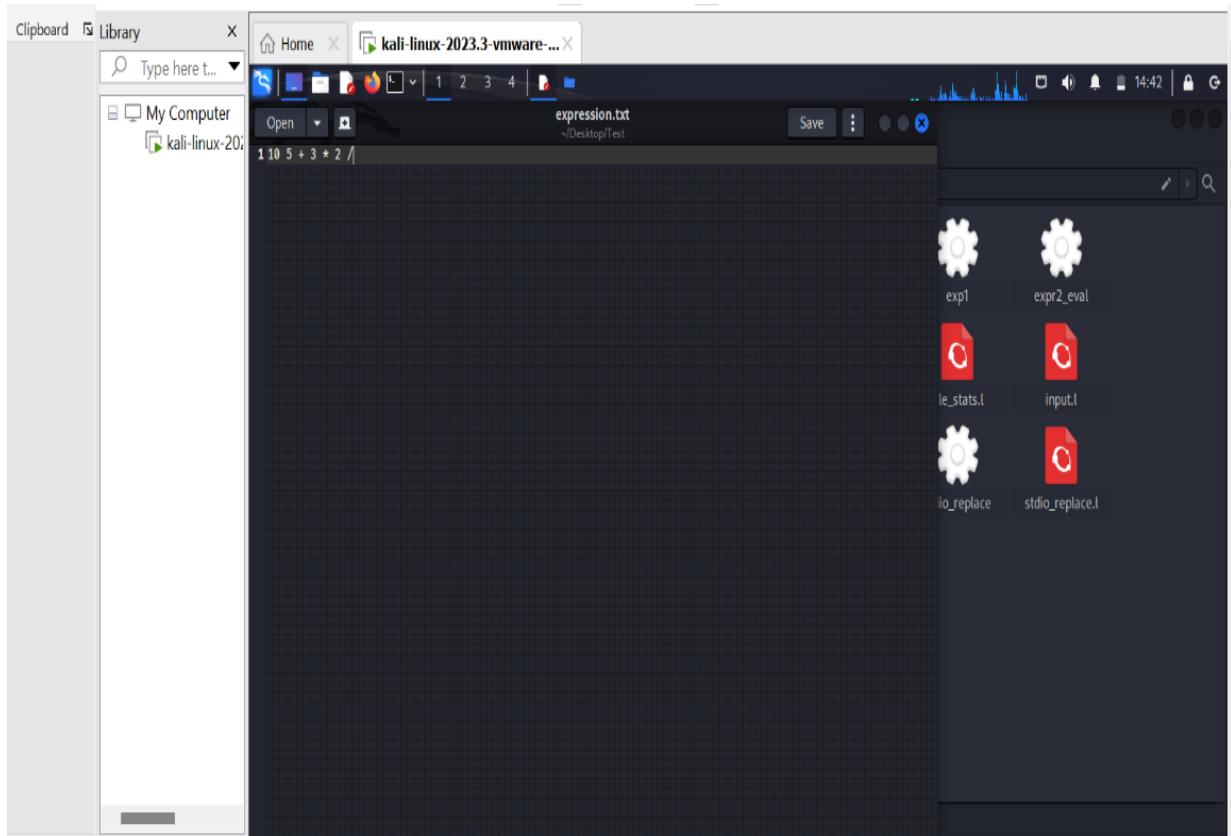
The file browser window shows a directory structure:

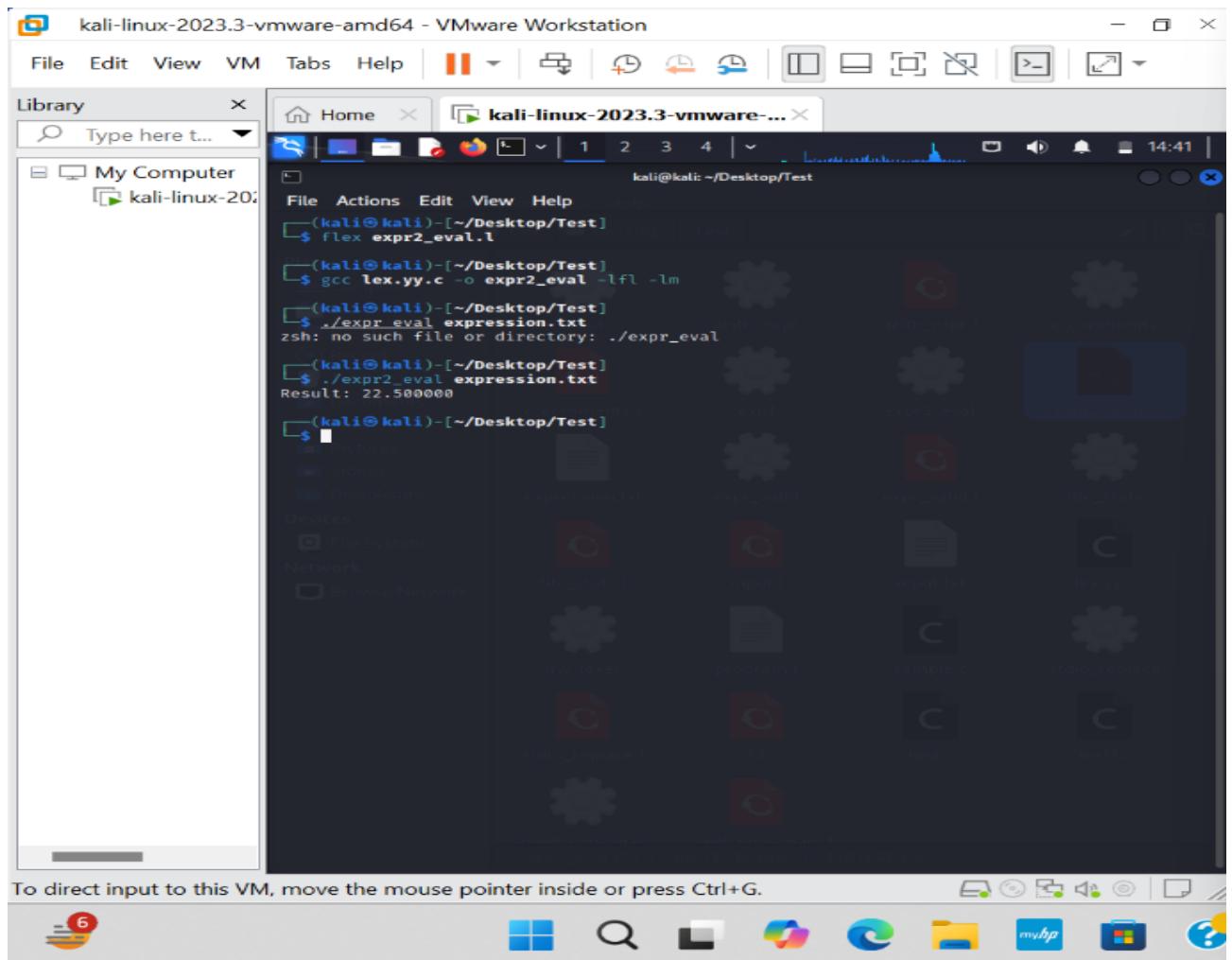
- My Computer
- kali-linux-20...

The terminal window also displays the command "expr2\_eval.l" and its details:

```
unusual construct unusual construct
"expr2_eval.l" | 1.7 Kib (1,710 bytes) | MATLAB file
```

To direct input to this VM, move the mouse pointer inside or press Ctrl+G.

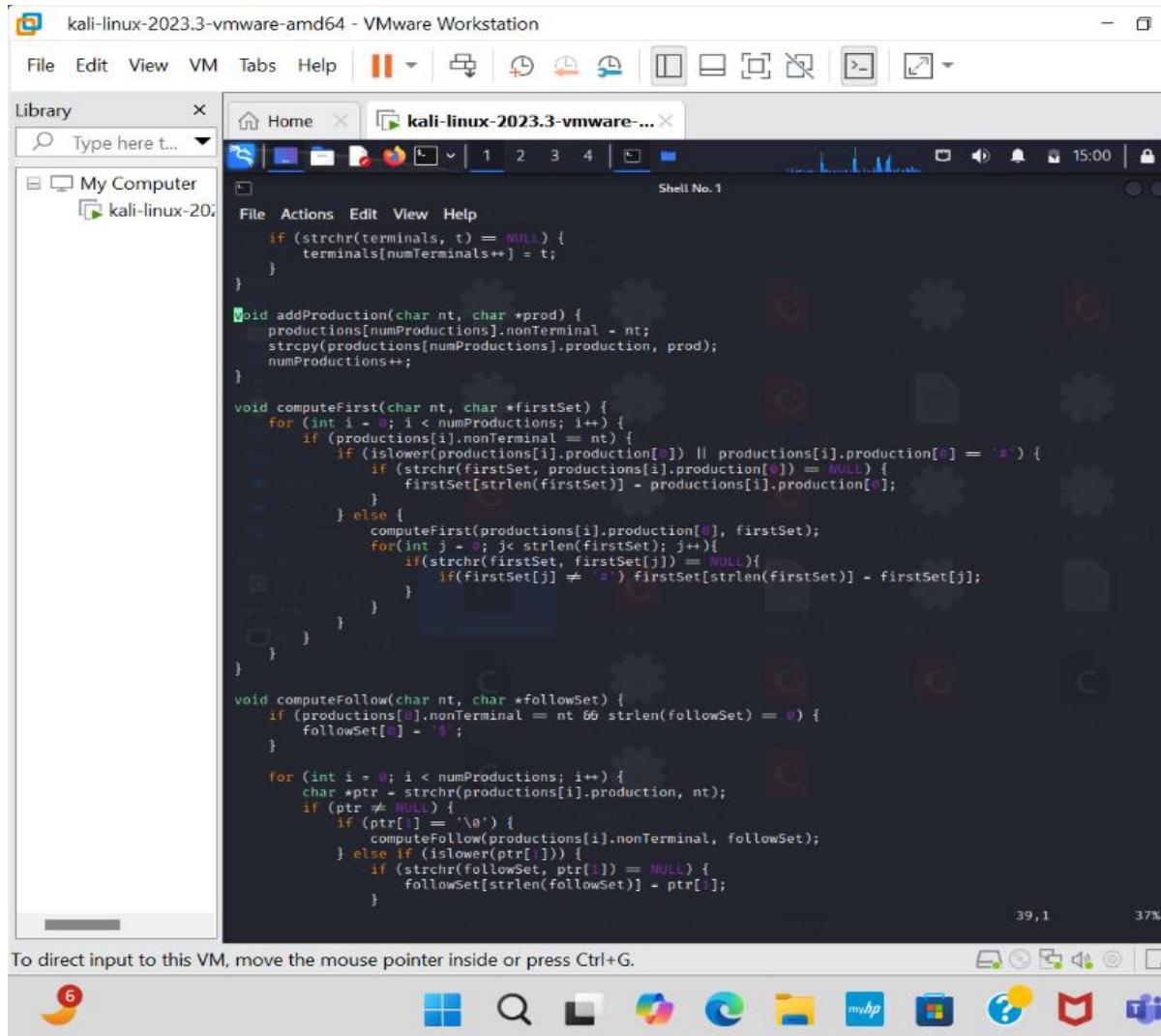




## EXPERIMENT NO-5

Title: Simulation of FIRST & FOLLOW of a Grammar.

1. WAP to simulate FIRST and FOLLOW of a grammar in C.



The screenshot shows a VMware Workstation window for a Kali Linux VM. The terminal window displays the following C code:

```
if (strchr(terminals, t) == NULL) {
    terminals[numTerminals++] = t;
}

void addProduction(char nt, char *prod) {
    productions[numProductions].nonTerminal = nt;
    strcpy(productions[numProductions].production, prod);
    numProductions++;
}

void computeFirst(char nt, char *firstSet) {
    for (int i = 0; i < numProductions; i++) {
        if (productions[i].nonTerminal == nt) {
            if (islower(productions[i].production[0]) || productions[i].production[0] == '#') {
                if (strchr(firstSet, productions[i].production[0]) == NULL) {
                    firstSet[strlen(firstSet)] = productions[i].production[0];
                }
            } else {
                computeFirst(productions[i].production[0], firstSet);
                for (int j = 0; j < strlen(firstSet); j++) {
                    if (strchr(firstSet, firstSet[j]) == NULL) {
                        if (firstSet[j] != '#') firstSet[strlen(firstSet)] = firstSet[j];
                    }
                }
            }
        }
    }
}

void computeFollow(char nt, char *followSet) {
    if (productions[0].nonTerminal == nt && strlen(followSet) == 0) {
        followSet[0] = '$';
    }

    for (int i = 0; i < numProductions; i++) {
        char *ptr = strchr(productions[i].production, nt);
        if (ptr != NULL) {
            if (ptr[1] == '\0') {
                computeFollow(productions[i].nonTerminal, followSet);
            } else if (islower(ptr[1])) {
                if (strchr(followSet, ptr[1]) == NULL) {
                    followSet[strlen(followSet)] = ptr[1];
                }
            }
        }
    }
}
```

The code implements functions to add productions, compute FIRST sets for non-terminals, and compute FOLLOW sets for non-terminals. It uses arrays to store terminals, productions, and sets of symbols.

kali-linux-2023.3-vmware-amd64 - VMware Workstation

File Edit View VM Tabs Help

Library Type here t... My Computer kali-linux-20...

Home kali-linux-2023.3-vmware-...

Shell No. 1

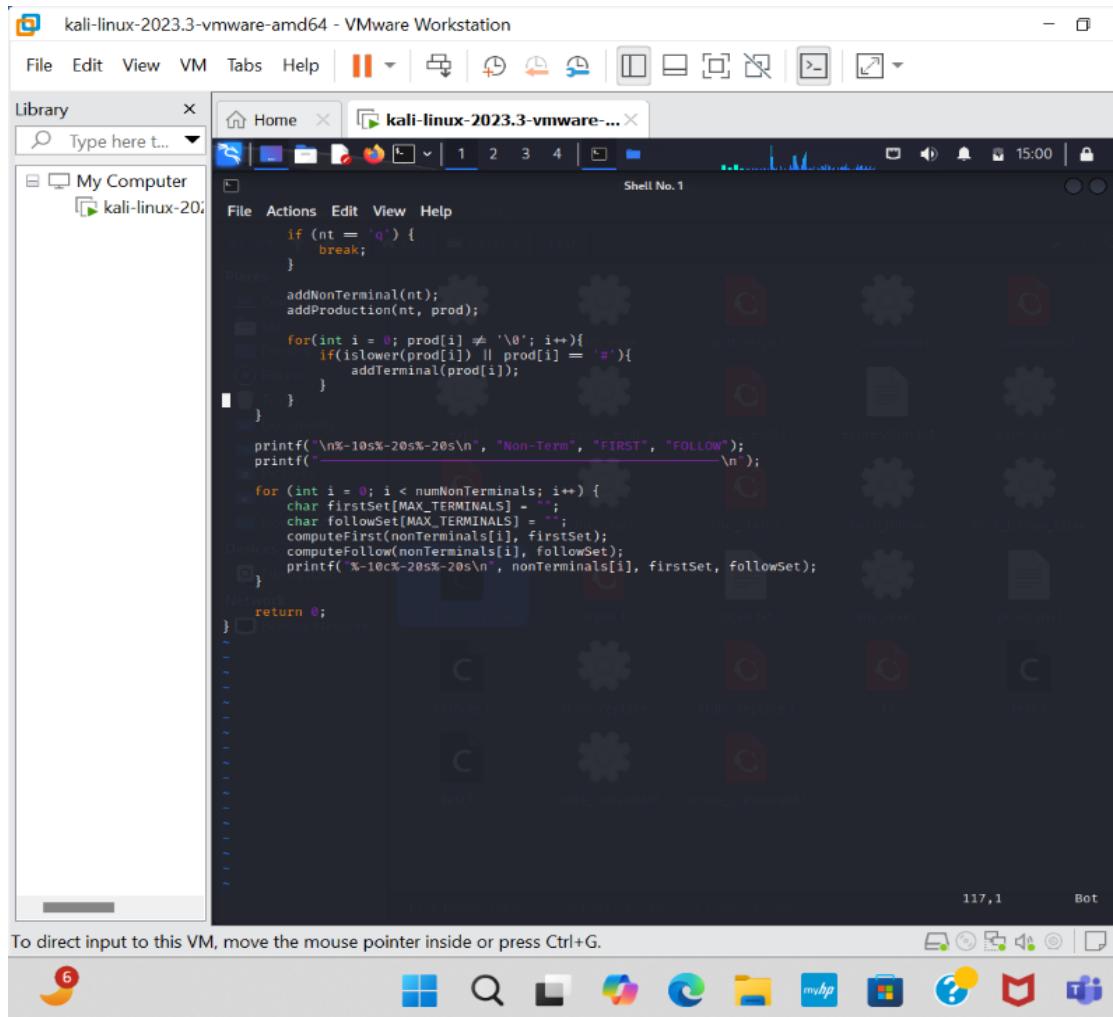
```
        computeFollow(productions[i].nonTerminal, followSet);
    } else if (islower(ptr[1])){
        if (strchr(followSet, ptr[1]) == NULL) {
            followSet[strlen(followSet)] = ptr[1];
        }
    } else {
        char firstSet[MAX_TERMINALS] = {};
        computeFirst(ptr[1], firstSet);
        for(int j = 0; j < strlen(firstSet); j++){
            if(strchr(followSet, firstSet[j]) == NULL || firstSet[j] != '#'){
                followSet[strlen(followSet)] = firstSet[j];
            }
        }
        if(strchr(firstSet,'#') != NULL){
            computeFollow(productions[i].nonTerminal, followSet);
        }
    }
}

int main() {
    initialize();
    int choice;
    char nt, prod[MAX_PRODUCTION_LEN];
    printf("Enter the grammar productions (enter 'q' to finish):\n");
    while (1) {
        printf("Enter non-terminal and production (e.g., S aABC):\n");
        if (scanf(" %c %s", &nt, prod) != 2) {
            break;
        }
        if (nt == 'q') {
            break;
        }
        addNonTerminal(nt);
        addProduction(nt, prod);
        for(int i = 0; prod[i] != '\0'; i++){
            if(islower(prod[i]) || prod[i] == '#'){
                addTerminal(prod[i]);
            }
        }
    }
}
```

78,1 81%

To direct input to this VM, move the mouse pointer inside or press Ctrl+G.

Windows Start button Taskbar icons



```
(kali㉿kali)-[~/Desktop/Test]
$ gcc first_follow_table.c -o first_follow_table

(kali㉿kali)-[~/Desktop/Test]
$ ./first_follow_table
Enter the grammar productions (enter 'q' to finish):
Enter non-terminal and production (e.g., S aAbc): S → aAbc
Enter non-terminal and production (e.g., S aAbc): Enter non-terminal and production (e.g., S aAbc): A → cB#
Enter non-terminal and production (e.g., S aAbc): Enter non-terminal and production (e.g., S aAbc): B → bAd
Enter non-terminal and production (e.g., S aAbc): Enter non-terminal and production (e.g., S aAbc): q
q

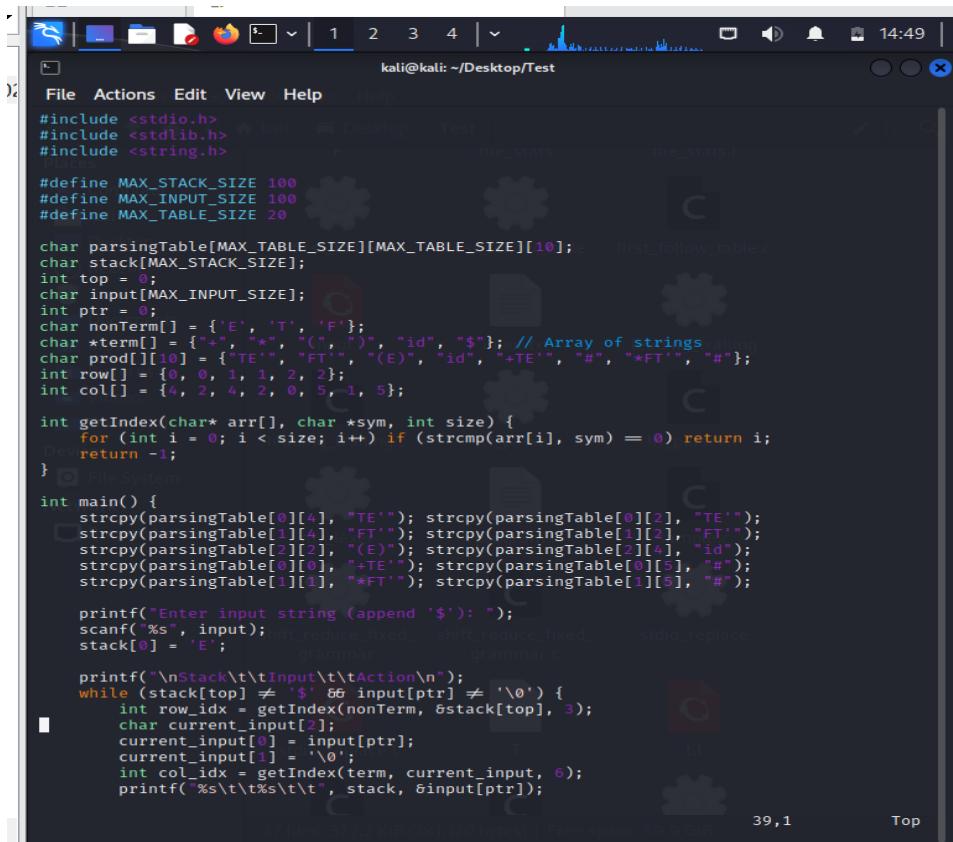
Non-Term FIRST FOLLOW
-----
S           $ 
a           d 
A           d 
c           c 
B           c 
b           c 

(kali㉿kali)-[~/Desktop/Test]
$ gcc first_follow_table.c -o first_follow_table

(kali㉿kali)-[~/Desktop/Test]
$ ./first_follow_table
Enter the grammar productions (enter 'q' to finish):
Enter non-terminal and production (e.g., S aAbc): 'c
```

## EXPERIMENT-6

### 1. WAP to construct LL(1) parser for an expression in C



The screenshot shows a terminal window titled "kali@kali: ~/Desktop/Test" with the following C code:

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

#define MAX_STACK_SIZE 100
#define MAX_INPUT_SIZE 100
#define MAX_TABLE_SIZE 20

char parsingTable[MAX_TABLE_SIZE][MAX_TABLE_SIZE][10];
char stack[MAX_STACK_SIZE];
int top = 0;
char input[MAX_INPUT_SIZE];
int ptr = 0;
char nonTerm[] = {'E', 'T', 'F'};
char *term[] = {"x", "*", "(", ")", "id", "$"}; // Array of strings
char prod[][10] = {"TE", "FT", "(E)", "id", "+TE", "#", "*FT", "#"};
int row[] = {0, 0, 1, 1, 2, 2};
int col[] = {4, 2, 4, 2, 0, 5, 1, 5};

int getIndex(char* arr[], char *sym, int size) {
    for (int i = 0; i < size; i++) if (strcmp(arr[i], sym) == 0) return i;
    return -1;
}

int main() {
    strcpy(parsingTable[0][4], "TE");
    strcpy(parsingTable[1][4], "FT");
    strcpy(parsingTable[2][4], "(E)");
    strcpy(parsingTable[0][0], "+TE");
    strcpy(parsingTable[0][5], "#");
    strcpy(parsingTable[1][1], "*FT");
    strcpy(parsingTable[1][5], "#");

    printf("\nEnter input string (append '$'): ");
    scanf("%s", input);
    stack[0] = 'E';
    printf("\nStack\tInput\tAction\n");
    while (stack[top] != '$' && input[ptr] != '\0') {
        int row_idx = getIndex(nonTerm, &stack[top], 3);
        char current_input[2];
        current_input[0] = input[ptr];
        current_input[1] = '\0';
        int col_idx = getIndex(term, current_input, 6);
        printf("%s\t%s\t", stack, &input[ptr]);
    }
}
```

```
kali@kali: ~/Desktop/Test
File Actions Edit View Help
int main() {
    strcpy(parsingTable[0][4], "TE'"); strcpy(parsingTable[0][2], "TE'");
    strcpy(parsingTable[1][4], "FT'"); strcpy(parsingTable[1][2], "FT'");
    strcpy(parsingTable[2][2], "(E)"); strcpy(parsingTable[2][4], "id");
    strcpy(parsingTable[0][0], "+TE'"); strcpy(parsingTable[0][5], "#");
    strcpy(parsingTable[1][1], "*FT'"); strcpy(parsingTable[1][5], "#");

    printf("\nEnter input string (append '$'): ");
    scanf("%s", input);
    stack[0] = 'E';

    printf("\nStack\tInput\tAction\n");
    while (stack[top] != '$' && input[ptr] != '\0') {
        int row_idx = getIndex(nonTerm, ostack[top], 3);
        char current_input[2];
        current_input[0] = input[ptr];
        current_input[1] = '\0';
        int col_idx = getIndex(term, current_input, 6);
        printf("%s\t%s\t", stack, &input[ptr]);

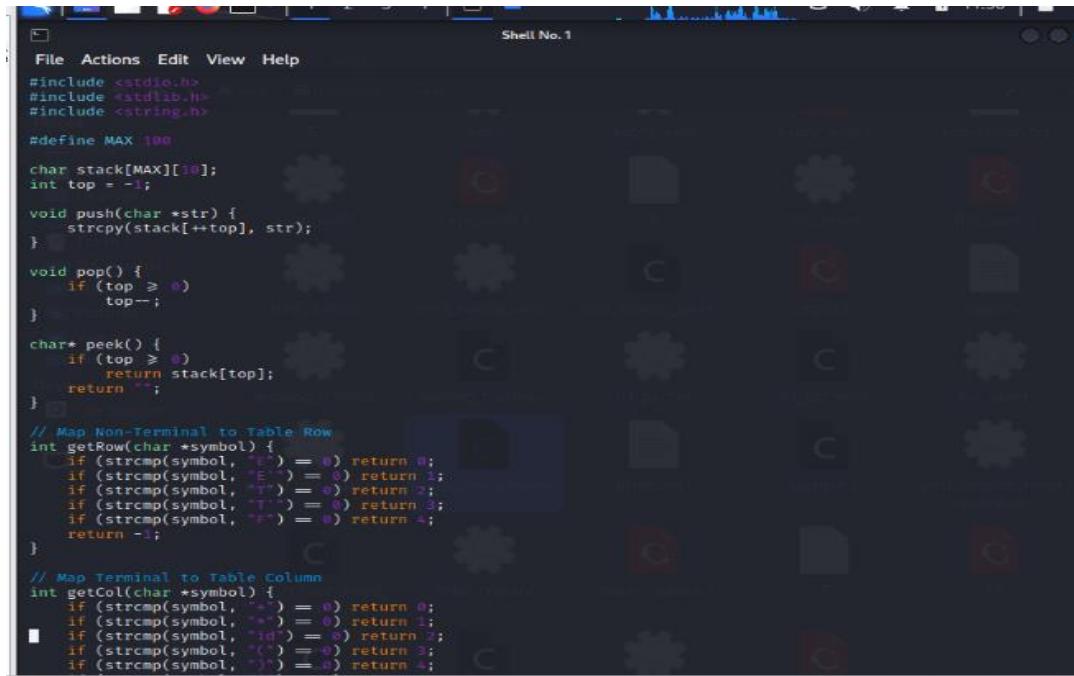
        if (stack[top] == input[ptr]) {
            printf("Match %c\n", stack[top--]);
            ptr++;
        } else if (row_idx != -1 && col_idx != -1 && parsingTable[row_idx][col_idx][0] != '\0') {
            printf("Apply %c -> %s\n", nonTerm[row_idx], parsingTable[row_idx][col_idx]);
            char *prod_str = parsingTable[row_idx][col_idx];
            top--;
            for (int i = strlen(prod_str) - 1; i >= 0; i--) {
                if (prod_str[i] != '#') stack[++top] = prod_str[i];
            }
            stack[top + 1] = '\0';
        } else {
            printf("Error\n"); break;
        }
    }
    if (stack[top] == '$' && input[ptr] == '\0') printf("\nParsing successful!\n");
    else printf("\nParsing failed!\n");
    return 0;
}

```

```
kali@kali: ~/Desktop/Test
File Actions Edit View Help
02
(kali㉿kali)-[~/Desktop/Test]
$ gcc lli_parser.c -o lli_parser
(kali㉿kali)-[~/Desktop/Test]
$ ./lli_parser
Enter tokens separated by space (e.g., id + id * id ), end with $:
> id + id * id $
✓ Input successfully parsed!
(kali㉿kali)-[~/Desktop/Test]
$
```

## EXPERIMENT-7

Title: Design of a Predictive Parser for a given Language Lab  
Activity: WAP to implement a Predictive Parser for a given language in C



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

#define MAX 100

char stack[MAX][10];
int top = -1;

void push(char *str) {
    strcpy(stack[++top], str);
}

void pop() {
    if (top >= 0)
        top--;
}

char* peek() {
    if (top >= 0)
        return stack[top];
    return "";
}

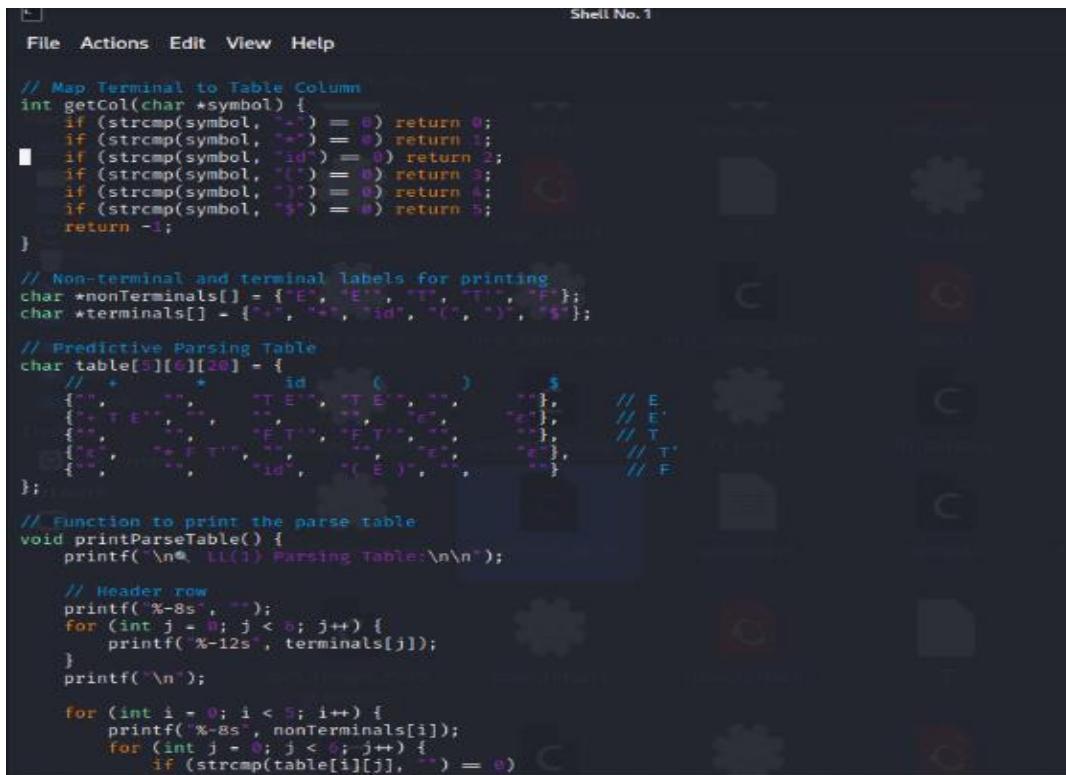
// Map Non-Terminal to Table Row
int getRow(char *symbol) {
    if (strcmp(symbol, "E") == 0) return 0;
    if (strcmp(symbol, "E'") == 0) return 1;
    if (strcmp(symbol, "T") == 0) return 2;
    if (strcmp(symbol, "T'") == 0) return 3;
    if (strcmp(symbol, "id") == 0) return 4;
    return -1;
}

// Map Terminal to Table Column
int getCol(char *symbol) {
    if (strcmp(symbol, "-") == 0) return 0;
    if (strcmp(symbol, "+") == 0) return 1;
    if (strcmp(symbol, "id") == 0) return 2;
    if (strcmp(symbol, "(") == 0) return 3;
    if (strcmp(symbol, ")") == 0) return 4;
    if (strcmp(symbol, "$") == 0) return 5;
    return -1;
}

// Non-terminal and terminal labels for printing
char *nonTerminals[] = {"E", "E'", "T", "T'", "F"};
char *terminals[] = {"-", "+", "id", "(", ")", "$"};

// Predictive Parsing Table
char table[5][6][20] = {
    // +   *   id   (   )   $
    { "", "T E", "T E'", "id", "()", "$"}, // E
    { "+ T E", "+", "T E'", "id", "()", "$"}, // E'
    { "", "T T'", "id", "()", "T'"}, // T
    { "T", "+ F T'", "id", "(", ")", "T'"}, // T'
    { "", "id", "id", "(", ")", "T'"}, // F
};

// Function to print the parse table
void printParseTable() {
    printf("\n%-11s Parsing Table:\n\n");
    // Header row
    printf("%-8s", "");
    for (int j = 0; j < 5; j++) {
        printf("%-12s", terminals[j]);
    }
    printf("\n");
    for (int i = 0; i < 5; i++) {
        printf("%-8s", nonTerminals[i]);
        for (int j = 0; j < 5; j++) {
            if (strcmp(table[i][j], "") == 0)
                printf(" ")
            else
                printf("%-12s", table[i][j]);
        }
        printf("\n");
    }
}
```



```
// Function to print the parse table
void printParseTable() {
    printf("\n%-11s Parsing Table:\n\n");
    // Header row
    printf("%-8s", "");
    for (int j = 0; j < 5; j++) {
        printf("%-12s", terminals[j]);
    }
    printf("\n");
    for (int i = 0; i < 5; i++) {
        printf("%-8s", nonTerminals[i]);
        for (int j = 0; j < 5; j++) {
            if (strcmp(table[i][j], "") == 0)
                printf(" ")
            else
                printf("%-12s", table[i][j]);
        }
        printf("\n");
    }
}
```

```
File Actions Edit View Help
    printf("\n");
}
printf("\n");
}

int main() {
    char input[MAX][10];
    int n = 0;

    // Print the parsing table
    printParseTable();

    printf("Enter the tokenized input (e.g., 3d + id + id $\n");
    while (!) {
        scanf("%s", input[n]);
        if (strcmp(input[n], "$") == 0)
            break;
        n++;
    }
    strcpy(input[++n], "$");
    push("$");
    push("E");

    int ip = 0;

    while (!) {
        char *X = peek();           // Stack top
        char *a = input[ip];       // Current input symbol

        if (
            strcmp(X, "+") == 0 || strcmp(X, "-") == 0 ||
            strcmp(X, "id") == 0 || strcmp(X, "(") == 0 ||
            strcmp(X, ")") == 0 || strcmp(X, "$") == 0
        ) {
            if (strcmp(X, a) == 0) {
                pop();
                ip++;
                if (strcmp(X, "$") == 0) {
                    printf("Input successfully parsed!\n");
                    break;
                }
            } else {
                printf("x Error: Mismatched symbol. Stack top: %s, Input: %s\n", X, a);
                break;
            }
        } else {
            int row = getRow(X);
            int col = getCol(a);

            if (row == -1 || col == -1 || strcmp(table[row][col], "") == 0) {
                printf("x Error: No rule for [%s, %s]\n", X, a);
                break;
            }

            pop();
            if (strcmp(table[row][col], "ε") != 0) {
                char production[20];
                strcpy(production, table[row][col]);

                char *symbols[s];
                int count = 0;
                char *token = strtok(production, " ");
                while (token != NULL) {
                    symbols[count++] = token;
                    token = strtok(NULL, " ");
                }

                for (int i = count - 1; i >= 0; i--) {
                    push(symbols[i]);
                }
            }
        }
    }
    return 0;
}
```

```
File Actions Edit View Help
    printf("x Input successfully parsed!\n");
    break;
} else {
    printf("x Error: Mismatched symbol. Stack top: %s, Input: %s\n", X, a);
    break;
} else {
    int row = getRow(X);
    int col = getCol(a);

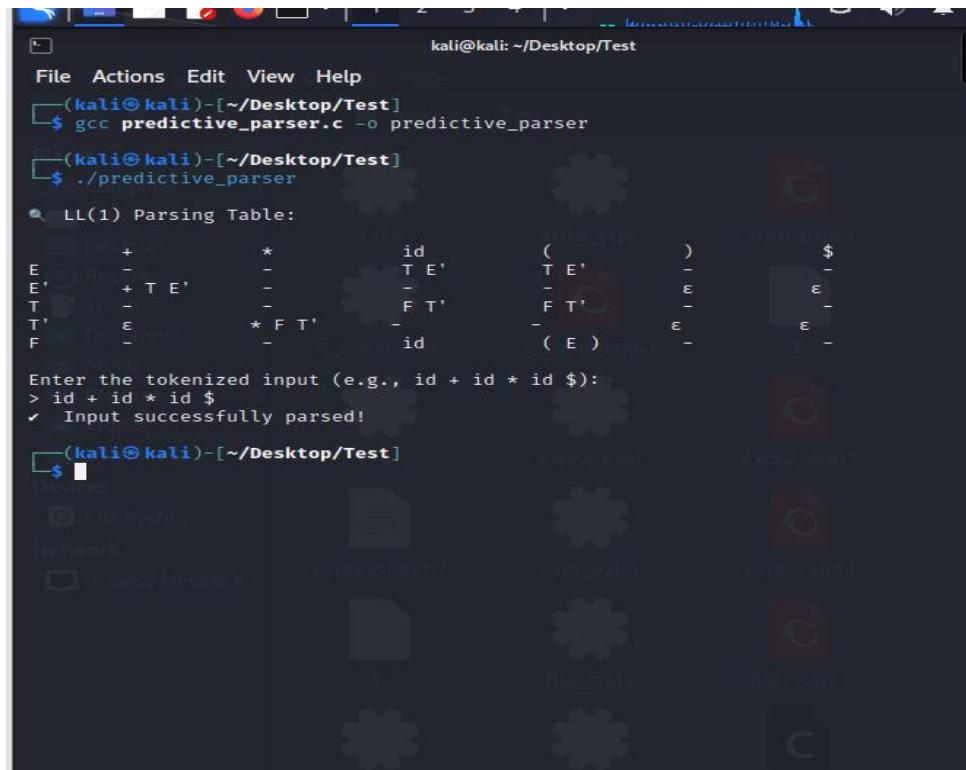
    if (row == -1 || col == -1 || strcmp(table[row][col], "") == 0) {
        printf("x Error: No rule for [%s, %s]\n", X, a);
        break;
    }

    pop();
    if (strcmp(table[row][col], "ε") != 0) {
        char production[20];
        strcpy(production, table[row][col]);

        char *symbols[s];
        int count = 0;
        char *token = strtok(production, " ");
        while (token != NULL) {
            symbols[count++] = token;
            token = strtok(NULL, " ");
        }

        for (int i = count - 1; i >= 0; i--) {
            push(symbols[i]);
        }
    }
}
return 0;
}
```

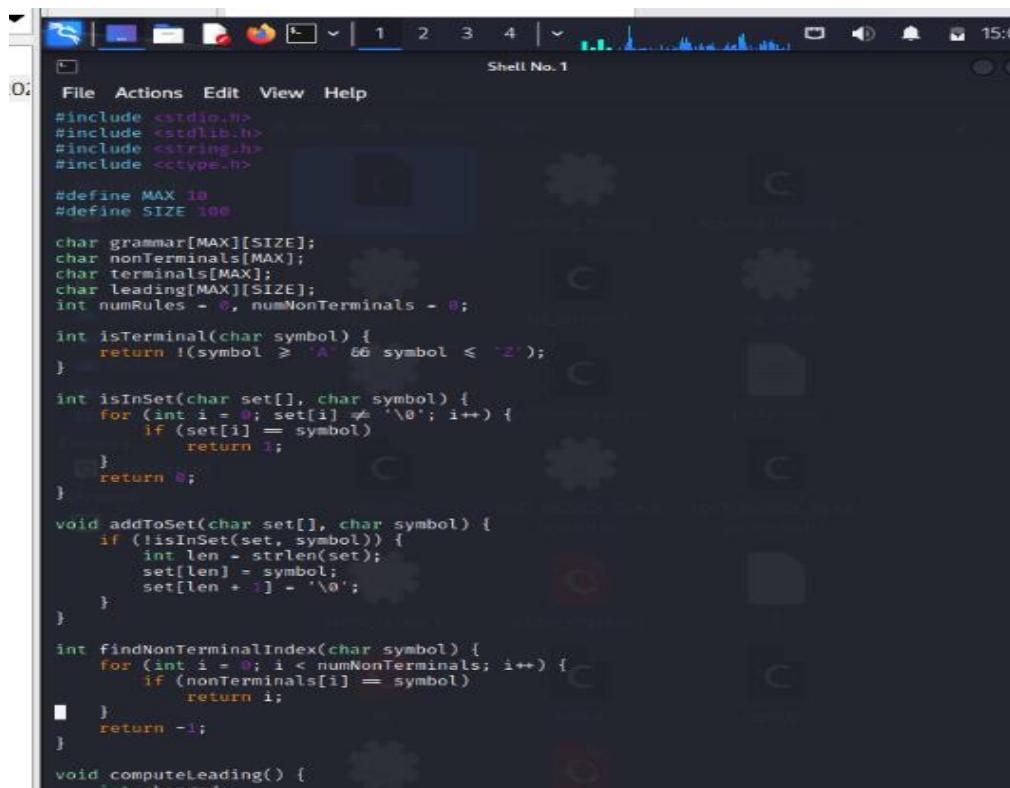
```
kali㉿kali: ~/Desktop/Test
File Actions Edit View Help
└─(kali㉿kali)-[~/Desktop/Test]
$ gcc predictive_parser.c -o predictive_parser
└─(kali㉿kali)-[~/Desktop/Test]
$ ./predictive_parser
e. LL(1) Parsing Table:
+      *      id      (      )      $
-      -      T E'    T E'    -      -
+ T E'  -      -      -      ε      ε
T      -      -      F T'    F T'    -      -
ε      * F T'  -      id     ( E )    ε      ε
F      -      -      -      -      -      -
Enter the tokenized input (e.g., id + id * id $):
> id + id * id $
✓ Input successfully parsed!
└─(kali㉿kali)-[~/Desktop/Test]
$
```



## **EXPERIMENT-8**

Title: Calculation of Leading for all the Non-Terminals Lab

Activity: WAP to calculate Leading for all the Non-Terminals in C



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

#define MAX 10
#define SIZE 100

char grammar[MAX][SIZE];
char nonTerminals[MAX];
char terminals[MAX];
char leading[MAX][SIZE];
int numRules = 0, numNonTerminals = 0;

int isTerminal(char symbol) {
    return !(symbol >= 'A' && symbol <= 'Z');
}

int isInSet(char set[], char symbol) {
    for (int i = 0; set[i] != '\0'; i++) {
        if (set[i] == symbol)
            return 1;
    }
    return 0;
}

void addToSet(char set[], char symbol) {
    if (!isInSet(set, symbol)) {
        int len = strlen(set);
        set[len] = symbol;
        set[len + 1] = '\0';
    }
}

int findNonTerminalIndex(char symbol) {
    for (int i = 0; i < numNonTerminals; i++) {
        if (nonTerminals[i] == symbol)
            return i;
    }
    return -1;
}

void computeLeading() {
    int changed;
```

```
Shell No.1
File Actions Edit View Help
changed = 0;
for (int i = 0; i < numRules; i++) {
    char lhs = grammar[i][0];
    char *rhs = strchr(grammar[i], '>') + 1;

    int lhsIndex = findNonTerminalIndex(lhs);

    if (isTerminal(rhs[0])) {
        if (!isInSet(leading[lhsIndex], rhs[0])) {
            addToSet(leading[lhsIndex], rhs[0]);
            changed = 1;
        }
    } else {
        int rhsIndex = findNonTerminalIndex(rhs[0]);
        for (int j = 0; leading[rhsIndex][j] != '\0'; j++) {
            if (!isInSet(leading[lhsIndex], leading[rhsIndex][j])) {
                addToSet(leading[lhsIndex], leading[rhsIndex][j]);
                changed = 1;
            }
        }

        if (rhs[1] != '\0' && isTerminal(rhs[1])) {
            if (!isInSet(leading[lhsIndex], rhs[1])) {
                addToSet(leading[lhsIndex], rhs[1]);
                changed = 1;
            }
        }
    }
}
} while (changed);

int main() {
    printf("Enter the number of productions: ");
    scanf("%d", &numRules);
    getchar(); // consume newline

    printf("Enter productions (e.g., E->TA)\n");
    for (int i = 0; i < numRules; i++) {
        fgets(grammar[i], SIZE, stdin);
        grammar[i][strcspn(grammar[i], "\n")] = '\0';
    }
}

54,1      66%
```

```
File Plugins Edit View Help
} while (changed);

int main() {
    printf("Enter the number of productions: ");
    scanf("%d", &numRules);
    getchar(); // consume newline

    printf("Enter productions (e.g., E->TA)\n");
    for (int i = 0; i < numRules; i++) {
        fgets(grammar[i], SIZE, stdin);
        grammar[i][strcspn(grammar[i], "\n")] = '\0';

        char lhs = grammar[i][0];
        if (!isInSet(nonTerminals, lhs)) {
            nonTerminals[numNonTerminals++] = lhs;
        }
    }

    computeLeading();

    printf("\n■ LEADING SETS:\n\n");
    printf("%-10s | %-10s\n", "Non-Terminal", "LEADING");
    printf("-----\n");

    for (int i = 0; i < numNonTerminals; i++) {
        printf("%-10c | %s\n", nonTerminals[i]);
        for (int j = 0; leading[i][j] != '\0'; j++) {
            printf("%c ", leading[i][j]);
        }
        printf("\n");
    }

    return 0;
}
```

kali@kali: ~/Desktop/Test

```
File Actions Edit View Help
[(kali㉿kali)-[~/Desktop/Test]]$ gcc leading.c -o leading
[(kali㉿kali)-[~/Desktop/Test]]$ ./leading
Enter the number of productions: 5
Enter productions (e.g., E→TA):
E→TA
A→+TA
A→ε
T→FB
F→id
expr1
expr2_eval
expr_valid
file_stats
first_follow
first_follow_table
first_follow
c_comments
c_comments.l
arith_expr
arith
expression.txt
F
c
[(kali㉿kali)-[~/Desktop/Test]]$
```

## EXPERIMENT-9

Title: Calculation of Trailing for all the Non-Terminals Lab

Activity: WAP to calculate Trailing for all the Non Terminals in C

```
1 #include <stdio.h>
2 #include <stdlib.h>
3 #include <string.h>
4 #include <ctype.h>
5
6 #define MAX 10
7 #define SIZE 100
8
9 char grammar[MAX][SIZE];
10 char nonTerminals[MAX];
11 char trailing[MAX][SIZE];
12 int numRules = 0, numNonTerminals = 0;
13
14 int isTerminal(char symbol) {
15     return !(symbol >= 'A' && symbol <= 'Z');
16 }
17
18 int isInSet(char set[], char symbol) {
19     for (int i = 0; set[i] != '\0'; i++) {
20         if (set[i] == symbol)
21             return 1;
22     }
23     return 0;
24 }
25
26 void addToSet(char set[], char symbol) {
27     if (!isInSet(set, symbol)) {
28         int len = strlen(set);
29         set[len] = symbol;
30         set[len + 1] = '\0';
31     }
32 }
33
34 int findNonTerminalIndex(char symbol) {
35     for (int i = 0; i < numNonTerminals; i++) {
36         if (nonTerminals[i] == symbol)
37             return i;
38     }
39     return -1;
40 }
41
42 void computeTrailing() {
43     int changed;
44     do {
45         changed = 0;
46
47         for (int i = 0; i < numRules; i++) {
48             char lhs = grammar[i][0];
49             char *rhs = strchr(grammar[i], '>') + 1;
50             int len = strlen(rhs);
51             int lhsIndex = findNonTerminalIndex(lhs);
52
53             // Case 1: If last symbol is terminal
54             if (isTerminal(rhs[len - 1])) {
55                 if (!isInSet(trailing[lhsIndex], rhs[len - 1])) {
56                     addToSet(trailing[lhsIndex], rhs[len - 1]);
57                     changed = 1;
58                 }
59             }
59             // Case 2: If last is non-terminal, copy its trailing
60             else {
61                 int ntIndex = findNonTerminalIndex(rhs[len - 1]);
62                 for (int j = 0; trailing[ntIndex][j] != '\0'; j++) {
63                     if (!isInSet(trailing[lhsIndex], trailing[ntIndex][j])) {
64                         addToSet(trailing[lhsIndex], trailing[ntIndex][j]);
65                         changed = 1;
66                     }
67                 }
68             }
69
70             // Check second last symbol, if terminal
71             if (len >= 2 && isTerminal(rhs[len - 2])) {
72                 if (!isInSet(trailing[lhsIndex], rhs[len - 2])) {
73                     addToSet(trailing[lhsIndex], rhs[len - 2]);
74                     changed = 1;
75                 }
76             }
77         }
78     } while (changed);
79 }
```

```

79     } while (changed);
80 }
81 }
82
83 int main() {
84     printf("Enter the number of productions: ");
85     scanf("%d", &numRules);
86     getchar(); // consume newline
87
88     printf("Enter productions (e.g., E→TA):\n");
89     for (int i = 0; i < numRules; i++) {
90         fgets(grammar[i], SIZE, stdin);
91         grammar[i][strcspn(grammar[i], "\n")] = '\0';
92
93         char lhs = grammar[i][0];
94         if (!isInSet(nonTerminals, lhs)) {
95             nonTerminals[numNonTerminals++] = lhs;
96         }
97     }
98
99     computeTrailing();
100
101    printf("\n■ TRAILING SETS:\n\n");
102    printf("%-10s | %-10s\n", "Non-Terminal", "TRAILING");
103    printf("-----\n");
104
105    for (int i = 0; i < numNonTerminals; i++) {
106        printf("%-10c | { ", nonTerminals[i]);
107        for (int j = 0; trailing[i][j] != '\0'; j++) {
108            printf("%c ", trailing[i][j]);
109        }
110        printf("}\n");
111    }
112
113    return 0;
114 }

```

er

-202

File Actions Edit View Help

(kali㉿kali)-[~/Desktop/Test]

\$ gcc trailing.c -o trailing

(kali㉿kali)-[~/Desktop/Test]

\$ ./trailing

Enter the number of productions: 5

Enter productions (e.g., E→TA):

E→TA  
A→+TA  
A→ε  
T→FB  
F→id

■ TRAILING SETS:

Non-Terminal	TRAILING
E	{ ◆ }
A	{ ◆ }
T	{ }
F	{ d }

(kali㉿kali)-[~/Desktop/Test]

\$

Devices

Network

Browse Network

firstFollow

firstFollowTable

firstFollowTable.c

leading

leadingTable

leadingTrailing

trailing

## EXPERIMENT-10

**Title:** Development of an Operator Precedence Parser Lab  
**Activity:** WAP to implement an Operator Precedence Parser for a given language in C



```
main.c
1 #include <stdio.h>
2 #include <string.h>
3 #include <ctype.h>
4
5 #define MAX 10
6
7 char nonTerminals[MAX][10], terminals[MAX][10], productions[MAX][50];
8 int numProductions = 0;
9
10 void getGrammar() {
11     printf("Enter number of productions: ");
12     scanf("%d", &numProductions);
13     printf("Enter productions:\n");
14     for (int i = 0; i < numProductions; i++) {
15         scanf("%s", productions[i]);
16     }
17 }
18
19 int isOperatorPrecedenceGrammar() {
20     for (int i = 0; i < numProductions; i++) {
21         if (strchr(productions[i], '|') != NULL) {
22             return 0;
23         }
24     }
25     return 1;
26 }
```

```
28+ void printLeadingTrailingSets() {
29+     printf("\n Leading and Trailing Sets\n");
30+     for (int i = 0; i < numProductions; i++) {
31+         printf("Leading(%c): { %id }%n", productions[i][@]);
32+         printf("Trailing(%c): { id,%} }%n", productions[i][@]);
33+     }
34+ }
35+
36+ void printFirstFollowSets() {
37+     printf("\n First and Follow Sets\n");
38+     for (int i = 0; i < numProductions; i++) {
39+         printf("First(%c): { %a }%n", productions[i][@]);
40+         printf("Follow(%c): { $ }%n", productions[i][@]);
41+     }
42+ }
43+
44+ void printOperatorPrecedenceTable() {
45+     printf("\nOperator Precedence Parsing Table\n");
46+     printf("t+\t*\\tid\t$\\n");
47+     printf(")*\\t>\\t<\\t<\\t>\\n");
48+     printf("*\\t>\\t<\\t<\\t>\\n");
49+     printf("id\\t>\\t<\\t<\\t>\\n");
50+     printf("$\\t>\\t<\\t<\\taccept\\n");
51+ }
52+
53+ void printPredictiveParsingTable() {
54+     printf("\nPredictive Parsing Table\\n");
55+ }
```

```
main.c
49     printf("id\t>\t>\t-\t>\n");
50     printf("$\t<\t<\t<\taccept\n");
51 }
52
53 void printPredictiveParsingTable() {
54     printf("\nPredictive Parsing Table\n");
55     printf("id\t+\t*\t$\n");
56     printf("E\tE->TE\tT\tT\n");
57     printf("E\tE->TE->TE\tT\tT\n");
58     printf("T\tT->FT\tT\tT\n");
59     printf("T\tT->E\tT\tT\n");
60     printf("FT\tFT->FT\n");
61 }
62
63 int main() {
64     getGrammar();
65
66     if (isOperatorPrecedenceGrammar()) {
67         printLeadingTrailingSets();
68         printOperatorPrecedenceTable();
69     } else {
70         printFirstFollowSets();
71         printPredictiveParsingTable();
72     }
73
74     return 0;
75 }
```

```
main.c
30     for (int i = 0; i < numProductions; i++) {
31         printf("Leading(%c): { (.id )}\n", productions[i][0]);
32         printf("Trailing(%c): { id,) }\n", productions[i][0]);
33     }
34 }
35
36 void printFirstFollowSets() {
37     printf("\n First and Follow Sets\n");
38     for (int i = 0; i < numProductions; i++) {
39         printf("First(%c): { a }\n", productions[i][0]);
40         printf("Follow(%c): { $ }\n", productions[i][0]);
41     }
42 }
43
44 void printOperatorPrecedenceTable() {
45     printf("\nOperator Precedence Table\n");
46
47     printf("Operator Precedence Parsing Table\n");
48     printf(" + * id $\n");
49     printf(" + > < < >\n");
50     printf(" * > > < >\n");
51     printf(" id > > - >\n");
52     printf(" $ < < < accept\n");
53     printf("♦ id\n");
54 }
```

input

```
Operator Precedence Parsing Table
+      *      id      $
+      >      <      <      >
*      >      >      <      >
id     >      >      -      >
$      <      <      <      accept
♦ id

...Program finished with exit code 0
Press ENTER to exit console.
```

## EXPERIMENT-11

Title: Implementation of Shift-Reduce Parser Lab Activity:  
WAP to implement Shift-Reduce Parser in C

```
1 #include <stdio.h>
2 #include <stdlib.h>
3 #include <string.h>
4
5 #define MAX 100
6 char input[MAX], stack[MAX];
7 int top = -1, ip = 0;
8 void printTable(const char *action) {
9     printf("%-20s %-20s %-20s\n", stack, input + ip, action);
10 }
11 int isReducible() {
12     /
13     if (top >= 1 && stack[top] == 'i' && stack[top - 1] == 'd')
14     {
15         stack[top - 1] = 'E';
16         top--;
17     } else if (top >= 2 && stack[top] == 'E' && stack[top - 1]
18             == '+' && stack[top - 2] == 'E') { // E + E -> E
```

```
17     }
18     top -= 2;
19     stack[top] = 'E';
20     return 1;
21 } else if (top >= 2 && stack[top] == 'E' && stack[top - 1]
22             == '*' && stack[top - 2] == 'E') { // E * E -> E
23     top -= 2;
24     stack[top] = 'E';
25     return 1;
26 } else if (top >= 2 && stack[top] == ')' && stack[top - 1]
27             == 'E' && stack[top - 2] == '(') { // (E) -> E
28     top -= 2;
29     stack[top] = 'E';
30     return 1;
31 }
32 }
```

```
33 ~ int main() {
34     printf("Enter input string (tokenized, end with $):\n> ");
35     scanf("%s", input);
36     printf("\n%-20s %-20s %-20s\n", "STACK", "INPUT", "ACTION");
37     printf("-----\n");
38
39 ~     while (1) {
40         stack[++top] = input[ip++];
41         stack[top + 1] = '\0';
42         printTable("SHIFT");
43 ~         while (isReducible()) {
44             printTable("REDUCE");
45         }
46 ~         if (stack[0] == 'E' && stack[1] == '\0' && input[ip] ==
47             '$') {
48             printTable("ACCEPTED");
49             break;
50     }
```

```
41         stack[top + 1] = '\0';
42         printTable("SHIFT");
43 ~         while (isReducible()) {
44             printTable("REDUCE");
45         }
46 ~         if (stack[0] == 'E' && stack[1] == '\0' && input[ip] ==
47             '$') {
48             printTable("ACCEPTED");
49             break;
50     }
51 ~         if (input[ip] == '\0') {
52             printTable("REJECTED");
53             break;
54     }
55
56     return 0;
57 }
```

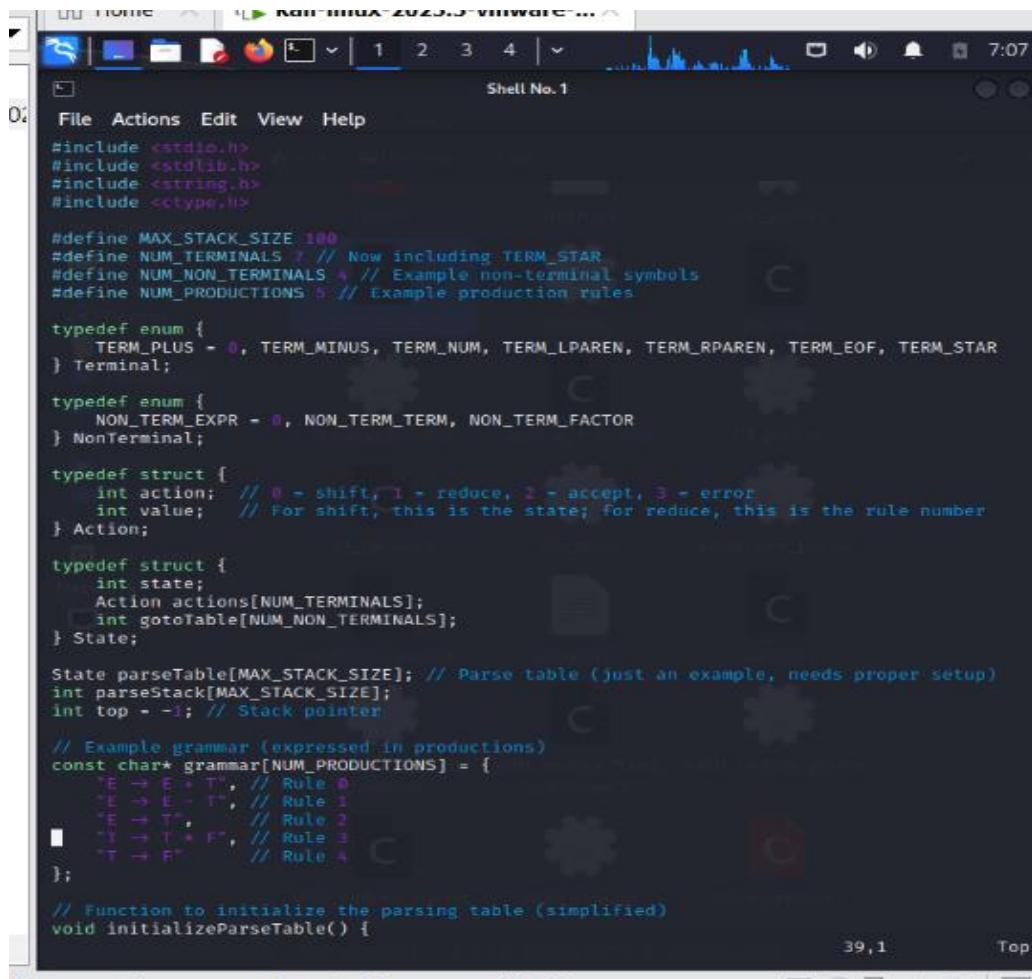
Enter input string (tokenized, end with \$):

> id+id\*id\$

STACK	INPUT	ACTION
i	d+id*id\$	SHIFT
id	+id*id\$	SHIFT
id+	id*id\$	SHIFT
id+i	d*id\$	SHIFT
id+id	*id\$	SHIFT
id+id*	id\$	SHIFT
id+id*i	d\$	SHIFT
id+id*id	\$	SHIFT
id+id*id\$		SHIFT
id+id*id\$		Accepted

## **EXPERIMENT-12**

Title: Design of a LALR Bottom-Up Parser for a given Language Lab Activity: WAP to derive a LALR Bottom Up Parser for the given language in C



The screenshot shows a terminal window titled "Shell No.1" with the following C code:

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

#define MAX_STACK_SIZE 100
#define NUM_TERMINALS 7 // Now including TERM_STAR
#define NUM_NON_TERMINALS 4 // Example non-terminal symbols
#define NUM_PRODUCTIONS 5 // Example production rules

typedef enum {
    TERM_PLUS = 0, TERM_MINUS, TERM_NUM, TERM_LPAREN, TERM_RPAREN, TERM_EOF, TERM_STAR
} Terminal;

typedef enum {
    NON_TERM_EXPR = 0, NON_TERM_TERM, NON_TERM_FACTOR
} NonTerminal;

typedef struct {
    int action; // 0 = shift, 1 = reduce, 2 = accept, 3 = error
    int value; // For shift, this is the state; for reduce, this is the rule number
} Action;

typedef struct {
    int state;
    Action actions[NUM_TERMINALS];
    int gotoTable[NUM_NON_TERMINALS];
} State;

State parseTable[MAX_STACK_SIZE]; // Parse table (just an example, needs proper setup)
int parseStack[MAX_STACK_SIZE];
int top = -1; // Stack pointer

// Example grammar (expressed in productions)
const char* grammar[NUM_PRODUCTIONS] = {
    "E → E + T", // Rule 0
    "E → E - T", // Rule 1
    "E → T", // Rule 2
    "T → T * F", // Rule 3
    "T → F" // Rule 4
};

// Function to initialize the parsing table (simplified)
void initializeParseTable() {
```

```
Home Kali-Linux-2023.3-vmware-... 7:08
File Actions Edit View Help
// Function to initialize the parsing table (simplified)
void initializeParseTable() {
    // Example: Just fill some values for demo purposes
    for (int i = 0; i < NUM_TERMINALS; i++) {
        for (int j = 0; j < MAX_STACK_SIZE; j++) {
            parseTable[j].actions[i].action = -3; // Default action is error
            parseTable[j].actions[i].value = -3;
        }
    }

    // Example: Define a simple shift action for some states
    parseTable[0].actions[TERM_NUM].action = 0; // Shift
    parseTable[0].actions[TERM_NUM].value = 1; // Transition to state 1
    parseTable[1].actions[TERM_PLUS].action = 1; // Reduce by rule 0
    parseTable[1].actions[TERM_MINUS].value = 0; // Rule number for reduction
    parseTable[1].actions[TERM_MINUS].action = 1; // Reduce by rule 1
    parseTable[1].actions[TERM_MINUS].value = 1; // Rule number for reduction
    parseTable[1].actions[TERM_EOF].action = 2; // Accept on EOF

    // Add missing action: handle TERM_NUM for state 3
    parseTable[1].actions[TERM_NUM].action = 0; // Shift
    parseTable[1].actions[TERM_NUM].value = 2; // Transition to state 2

    // Handle other terminal symbols for state transitions (add cases as needed)
    parseTable[2].actions[TERM_PLUS].action = 1; // Reduce by rule 2
    parseTable[2].actions[TERM_PLUS].value = 2; // Rule number for reduction
    parseTable[2].actions[TERM_MINUS].action = 1; // Reduce by rule 3
    parseTable[2].actions[TERM_MINUS].value = 3; // Rule number for reduction
    parseTable[2].actions[TERM_EOF].action = 2; // Accept on EOF
}

// Simple function to tokenize input into terminal symbols
Terminal getToken(const char **input) {
    while (**input == ' ' || **input == '\t') { // Skip whitespace
        (*input)++;
    }

    if (isdigit(**input)) { // If it's a number
        printf("Token: NUM\n"); // Debug output
        return TERM_NUM;
    }

    char currentChar = **input;
    (*input)++;
}
```

```
Shell No.1
File Actions Edit View Help
char currentChar = **input;
(*input)++;

switch (currentChar) {
    case '+':
        printf("Token: PLUS\n"); // Debug output
        return TERM_PLUS;
    case '-':
        printf("Token: MINUS\n"); // Debug output
        return TERM_MINUS;
    case '*':
        printf("Token: STAR\n"); // Debug output
        return TERM_STAR; // Corrected here
    case '(':
        printf("Token: LPAREN\n"); // Debug output
        return TERM_LPAREN;
    case ')':
        printf("Token: RPAREN\n"); // Debug output
        return TERM_RPAREN;
    case '\0':
        printf("Token: EOF\n"); // Debug output
        return TERM_EOF;
    default:
        printf("Unknown Token: %c\n", currentChar); // Debug output for unknown chars
        return TERM_EOF;
}

// Function to perform a shift operation
void shift(int state){
    parseStack[++top] = state;
}

// Function to perform a reduce operation
void reduce(int rule) {
    top--; // Pop state(s) from the stack based on the right-hand side of the rule
}

// Main function to simulate a simple bottom-up parser
void parse(const char* input) {
    int state = 0; // Initial state
    int i = 0; // Input index
    const char *input_ptr = input;
```

90,1

52%

```
int step = 1; // Step counter
while (1) {
    Terminal currentTerminal = getToken(input_ptr); // Get the next token
    printf("Current Token: %d\n", currentTerminal); // Debug output

    // Print the current status
    printf("%-5d", step); // Print the step
    printf("%-10s", ""); // State stack (we'll print it later)
    printf("%-10d", currentTerminal); // Current token (for debugging)

    Action currentAction = parseTable[state].actions[currentTerminal];
    printf("State: %d, Action: %d, Value: %d\n", state, currentAction.action, currentAction.value); // Debug output

    if (currentAction.action == 0) {
        // Shift operation
        state = currentAction.value; // Use the state value from the action
        shift(state);
        printf("Shift   %-10d\n", state);
    } else if (currentAction.action == 1) {
        // Reduce operation
        int rule = currentAction.value;
        reduce(rule);
        state = parseStack[top]; // Pop and transition to the previous state
        printf("Reduce   %-10d\n", state);
    } else if (currentAction.action == 2) {
        // Accept operation
        printf("Accept   \n");
        break;
    } else {
        printf("Accept   \n");
        break;
    }
    step++;
}
printf("\n");
}
```

```
File Actions Edit View Help
    state = parseStack[top]; // Pop and transition to the previous state
}
else if (currentAction.action == 2) {
    // Accept operation
    printf("Accept | %-10d |\n", state);
    break;
}
else {
    printf("Accept | |\n");
    break;
}
step++;
}
printf("-----+\n");
}
}

int main() {
    initializeParseTable(); // Set up the parse table (to be filled with correct values)

    // Get input from the user
    char input[100];
    printf("Enter an expression to parse (e.g., 3+4*5): ");
    fgets(input, sizeof(input), stdin);
    input[strcspn(input, "\n")] = '\0'; // Remove newline character if present

    parse(input); // Parse the input

    return 0;
}
```

kali@kali: ~/Desktop/Test

File Actions Edit View Help Help

```
(kali㉿kali)-[~/Desktop/Test]
$ gcc lalr_parser.c -o lalr_parser
```

```
(kali㉿kali)-[~/Desktop/Test]
$ ./lalr_parser
Enter an expression to parse (e.g., 3+4*5): 4+3*5
```

Step	State Stack	Current	Action	Next State
1	1	2	State: 0, Action: 0, Value: 1	
	Shift	1		c_comments
Token: NUM				
Current Token: 2				
2	2	2	State: 1, Action: 0, Value: 2	
	Shift	2		c_comments
Token: NUM				
Current Token: 2				
3	3	2	State: 2, Action: 3, Value: -1	
Accept				

```
$
```

File System

expression.txt expr\_valid file\_stats file\_stats.l first\_follow first\_follow\_table first\_follow\_table.c

Log Out...