



CSM 3202: Compiler Lab

Lab Report 1: Lexical Analysis and Token Classification

Name: MD. TAMIM AHMED FAHIM

ID: 2209024

Level: 3, Semester: 2

Bioinformatics Engineering

Bangladesh Agricultural University

Report submitted: December 2, 2025

This is a documentation to the Lab X done in the Compiler Lab Course. The report is submitted to Md. Saif Uddin, Lecturer, Department of Computer Science and Mathematics, Bangladesh Agricultural University.

Contents

1	Introduction	4
1.1	Background of the Experiment	4
1.2	Objectives of the Lab	4
1.3	Tools and Requirements	4
2	Tasks and Implementations	5
2.1	Task 1: Characters, Words, and Lines Count	5
2.1.1	Task Description	5
2.1.2	Source Code	5
2.1.3	Input & Output	6
2.1.4	Explanation of Code	7
2.2	Task 2: Identify Keywords	8
2.2.1	Task Description	8
2.2.2	Source Code	8
2.2.3	Input & Output	10
2.2.4	Explanation of Code	10
2.3	Task 3: Identification of Valid and Invalid Identifiers	11
2.3.1	Task Description	11
2.3.2	Source Code	11
2.3.3	Input & Output	13
2.3.4	Explanation of Code	13
2.4	Task 4: Identification of Numeric Constants	14
2.4.1	Task Description	14
2.4.2	Source Code	14
2.4.3	Input & Output	16
2.4.4	Explanation of Code	16
2.5	Task 5: Identification of Operators	17
2.5.1	Task Description	17
2.5.2	Source Code	17
2.5.3	Input & Output	20
2.5.4	Explanation of Code	20
2.6	Task 6: Identification of Punctuation and Special Symbols	21
2.6.1	Task Description	21
2.6.2	Source Code	21
2.6.3	Input & Output	22
2.6.4	Explanation of Code	22
2.7	Task 7: Identification of Single-line and Multi-line Comments	23
2.7.1	Task Description	23
2.7.2	Source Code	23

2.7.3	Input & Output	24
2.7.4	Explanation of Code	25
2.8	Task 8: Identification of String Literals and Character Constants	26
2.8.1	Task Description	26
2.8.2	Source Code	26
2.8.3	Input & Output	28
2.8.4	Explanation of Code	28
2.9	Task 9: Full Lexical Token Classification and Token Stream Generation	29
2.9.1	Task Description	29
2.9.2	Source Code	29
2.9.3	Input & Output	35
2.9.4	Explanation of Code	35
2.10	Task 10: Detection of Lexical Errors	36
2.10.1	Task Description	36
2.10.2	Source Code	36
2.10.3	Input & Output	40
2.10.4	Explanation of Code	40
3	Conclusion	41

List of Figures

List of Tables

Listings

1	A C program to count characters, words, and lines	5
2	A C program to Identify Keywords	8
3	A C program to Identification of Valid and Invalid Identifiers	11
4	A C program to identification of Numeric Constants	14
5	A C program to Identification of Operators	17
6	A C program to Identification of Punctuation and Special Symbols	21
7	A C program to Identification of Single-line and Multi-line Comments	23
8	A C program to Identification of String Literals and Character Constants	26

9	A C program for Full Lexical Token Classification and Token Stream Generation	29
10	A C program to Detection of Lexical Errors	36

1 Introduction

1.1 Background of the Experiment

You will explain what lexical analysis is, why it is used in compiler design, and the overall purpose of token identification.

1.2 Objectives of the Lab

You will list 3–5 clear objectives describing what you aim to learn, such as writing lex programs, identifying tokens, and classifying input streams.

1.3 Tools and Requirements

You will list the software used (e.g. IDE, C Compiler, Windows/Ubuntu OS, terminal/command prompt).

2 Tasks and Implementations

2.1 Task 1: Characters, Words, and Lines Count

2.1.1 Task Description

This program reads a source file sequentially and counts the total number of characters, words, and lines. It correctly handles spaces, tabs, and newline characters while scanning the file.

2.1.2 Source Code

```
1  #include <stdio.h>
2  #include <stdlib.h>
3  #include <ctype.h>
4
5  int main()
6  {
7
8      FILE *fp = NULL;
9      char ch;
10     int character = 0;
11     int word = 0;
12     int line = 0;
13     int word_flag = 0;
14
15     fp = fopen("task1.txt", "r");
16     if (fp == NULL)
17     {
18         printf("file not found");
19         exit(1);
20     }
21
22     // reading character by character
23     while ((ch = fgetc(fp)) != EOF)
24     {
25         character++;
26         if (ch == '\n')
27         {
28             line++;
29         }
```

```

30
31     if (isspace(ch))
32     {
33         word_flag = 1;
34     }
35     else if (word_flag == 1)
36     {
37         word_flag = 0;
38         word++;
39     }
40
41     // printf("%c ",ch);
42 }
43
44 word++;
45
46 fclose(fp);
47 printf("Characters:␣%d\n", character);
48 printf("Words:␣%d\n", word);
49 printf("Lines:␣%d\n", line);
50
51 return 0;
52 }

```

Listing 1: A C program to count characters, words, and lines

2.1.3 Input & Output

You will write the input sample you provided to the program exactly as executed and insert the screenshot of your output terminal here. Make sure the image is clear and readable.

Input:

```

1 int main() {
2     printf("Hello World");
3 }
4

```

Output:

```
Characters: 42  
Words: 6  
Lines: 3
```

2.1.4 Explanation of Code

The program read each character from file and calculate character, whhen it find ' ' or '\t' or '\n' or other whitespace it count word and when it fine '\n' it count new line.

2.2 Task 2: Identify Keywords

2.2.1 Task Description

This program takes input line of C code, identify and print all the keywords present in the line. Non-keyword identifiers should not be printed.

2.2.2 Source Code

```
1 #include <stdio.h>
2 #include <stdlib.h>
3 #include <ctype.h>
4 #include <string.h>
5
6 int is_keyword(const char *word)
7 {
8     const char *keywords[] = {
9         "auto", "break", "case", "char", "const", "
10         continue", "default", "do", "double",
11         "else", "enum", "extern", "float", "for", "
12         goto", "if", "inline", "int", "long",
13         "register", "restrict", "return", "short", "
14         signed", "sizeof", "static",
15         "struct", "switch", "typedef", "union", "
16         unsigned", "void", "volatile", "while"};
17
18     int n = sizeof(keywords) / sizeof(keywords[0]);
19     for (int i = 0; i < n; i++)
20     {
21         if (strcmp(word, keywords[i]) == 0)
22             return 1;
23     }
24     return 0;
25 }
26
27 int main()
28 {
29     FILE *fp = NULL;
30     char ch;
31     char new_word[40];
32     int index = 0;
```

```

30
31     fp = fopen("task2.txt", "r");
32     if (fp == NULL)
33     {
34         printf("file not found");
35         exit(1);
36     }
37
38     while ((ch = fgetc(fp)) != EOF)
39     {
40         if (isalpha(ch))
41         {
42             index = 0;
43             new_word[index] = ch;
44             index++;
45
46             while ((ch = fgetc(fp)) != EOF && isalpha(
47                 ch))
48             {
49                 new_word[index] = ch;
50                 index++;
51             }
52
53             new_word[index] = '\0';
54
55             // printf("%s@ ", new_word);
56
57             if (is_keyword(new_word))
58             {
59                 printf("%s: keyword\n", new_word);
60             }
61         }
62
63     }
64
65     fclose(fp);
66
67     return 0;
68 }

```

Listing 2: A C program to Identify Keywords

2.2.3 Input & Output

Input:

```
1 int main() {  
2     float num = 2.5;  
3     return 0;  
4 }
```

Output:

```
int : keyword  
float : keyword  
return : keyword
```

2.2.4 Explanation of Code

Firstly each word is identified and a function is used to check if that word in keyword array.

2.3 Task 3: Identification of Valid and Invalid Identifiers

2.3.1 Task Description

A program to extract all identifiers from a given line and classify them as valid or invalid.

2.3.2 Source Code

```
1  #include <stdio.h>
2  #include <stdlib.h>
3  #include <ctype.h>
4  #include <string.h>
5
6  int is_identifier(const char *word)
7  {
8      const char *keywords[] = {
9          "auto", "break", "case", "char", "const", "
10         continue", "default", "do", "double",
11         "else", "enum", "extern", "float", "for", "
12         goto", "if", "inline", "int", "long",
13         "register", "restrict", "return", "short", "
14         signed", "sizeof", "static",
15         "struct", "switch", "typedef", "union", "
16         unsigned", "void", "volatile", "while"};
17
18     if (isdigit(word[0]))
19     {
20         return 0;
21     }
22
23     int n = sizeof(keywords) / sizeof(keywords[0]);
24     for (int i = 0; i < n; i++)
25     {
26         if (strcmp(word, keywords[i]) == 0)
27             return 0;
28     }
29     return 1;
30 }
```

```

28 int main()
29 {
30
31     FILE *fp = NULL;
32     char ch;
33     char new_word[40];
34     int index = 0;
35
36     fp = fopen("task3.txt", "r");
37     if (fp == NULL)
38     {
39         printf("file_not_found");
40         exit(1);
41     }
42
43     while ((ch = fgetc(fp)) != EOF)
44     {
45         if (isalnum(ch) || ch == '_')
46         {
47             index = 0;
48             new_word[index] = ch;
49             index++;
50
51             while ((ch = fgetc(fp)) != EOF && (isalnum
52                 (ch) || ch == '_'))
53             {
54                 new_word[index] = ch;
55                 index++;
56             }
57
58             new_word[index] = '\0';
59
60             // printf("%s@ ", new_word);
61
62             if (is_identifier(new_word) == 1)
63             {
64                 printf("%s: Valid Identifier\n",
65                     new_word);
66             }
67         }
68         else
69         {
70

```

```

67         printf("%s: Invalid Identifier\n",
68                new_word);
69     }
70 }
71
72 fclose(fp);
73
74 return 0;
75 }

```

Listing 3: A C program to Identification of Valid and Invalid Identifiers

2.3.3 Input & Output

Input:

```
1 id1 , _id2 , 3id , id 5
```

Output:

```

id1 : Valid Identifier
_id2 : Valid Identifier
3id : Invalid Identifier
id : Valid Identifier
5 : Invalid Identifier

```

2.3.4 Explanation of Code

Firstly identified each word and check with conditions of identifier in a function.

2.4 Task 4: Identification of Numeric Constants

2.4.1 Task Description

A program to detect and classify numeric constants into integer and floating point.

2.4.2 Source Code

```
1  #include <stdio.h>
2  #include <stdlib.h>
3  #include <ctype.h>
4  #include <string.h>
5
6  int is_number(const char *word)
7  {
8      int flag = 0;
9      int i = 0;
10     while (word[i] != '\0')
11     {
12         if (word[i] == '.')
13         {
14             flag++;
15         }
16         i++;
17     }
18     if (flag > 1)
19     {
20         return 0;
21     }
22     else if (flag == 1)
23     {
24         return 1;
25     }
26     else
27     {
28         return 2;
29     }
30 }
31
32 int main()
33 {
```

```

34
35 FILE *fp = NULL;
36 char ch;
37 char new_word[40];
38 int index = 0;
39
40 fp = fopen("task4.txt", "r");
41 if (fp == NULL)
42 {
43     printf("file not found");
44     exit(1);
45 }
46
47 while ((ch = fgetc(fp)) != EOF)
48 {
49     if (isdigit(ch) || ch == '.')
50     {
51         index = 0;
52         new_word[index] = ch;
53         index++;
54
55         while ((ch = fgetc(fp)) != EOF && (isdigit
56             (ch) || ch == '.'))
57         {
58             new_word[index] = ch;
59             index++;
60         }
61
62         new_word[index] = '\0';
63
64         // printf("%s@ ", new_word);
65
66         if (is_number(new_word) == 1)
67         {
68             printf("%s: Float\n", new_word);
69         }
70         else if (is_number(new_word) == 2)
71         {
72             printf("%s: Integer\n", new_word);
73         }
74         else

```



```

74         {
75             printf("%s : Not a number\n",
76                   new_word);
77         }
78     }
79
80     fclose(fp);
81
82     return 0;
83 }

```

Listing 4: A C program to identification of Numeric Constants

2.4.3 Input & Output

Input:

```
1 a = 10 ; b = 2.27 ; c = 300 ;
```

Output:

```

10 : Integer
2.27 : Float
300 : Integer

```

2.4.4 Explanation of Code

Firstly all word which are similar like digit is seperated then the are check through a user defined function.

2.5 Task 5: Identification of Operators

2.5.1 Task Description

A program to identify all assignment, arithmetic, relational, and logical operators.

2.5.2 Source Code

```
1  #include <stdio.h>
2
3  int main()
4  {
5      FILE *fp = fopen("task5.txt", "r");
6      if (fp == NULL)
7      {
8          printf("File not found\n");
9          return 1;
10     }
11
12     char ch, next_ch;
13
14     while ((ch = fgetc(fp)) != EOF)
15     {
16         int next_ch = fgetc(fp);
17         if (next_ch != EOF)
18         {
19             // Check 2-character operators
20             if (ch == '+' && next_ch == '+')
21             {
22                 printf("++: arithmetic operator\n");
23                 continue;
24             }
25             else if (ch == '-' && next_ch == '-')
26             {
27                 printf("--: arithmetic operator\n");
28                 continue;
29             }
30             else if (ch == '=' && next_ch == '=')
31             {
32                 printf("==: relational operator\n");
33                 continue;
```

```

34     }
35     else if (ch == '!' && next_ch == '=')
36     {
37         printf("!=:_:relational_operator\n");
38         continue;
39     }
40     else if (ch == '>' && next_ch == '=')
41     {
42         printf(">=_:relational_operator\n");
43         continue;
44     }
45     else if (ch == '<' && next_ch == '=')
46     {
47         printf("<=_:relational_operator\n");
48         continue;
49     }
50     else if (ch == '&' && next_ch == '&')
51     {
52         printf("&&:_:logical_operator\n");
53         continue;
54     }
55     else if (ch == '|' && next_ch == '|')
56     {
57         printf("||:_:logical_operator\n");
58         continue;
59     }
60     else if (ch == '+' && next_ch == '=')
61     {
62         printf("+=_:assignment_operator\n");
63         continue;
64     }
65     else if (ch == '-' && next_ch == '=')
66     {
67         printf("-=_:assignment_operator\n");
68         continue;
69     }
70     else if (ch == '*' && next_ch == '=')
71     {
72         printf("*=_:assignment_operator\n");
73         continue;
74     }

```

```

75         else if (ch == '/' && next_ch == '=')
76         {
77             printf("/=:assignment_operator\n");
78             continue;
79         }
80         else
81         {
82             ungetc(next_ch, fp);
83         }
84     }
85
86     // Single-character operators
87     if (ch == '=')
88         printf("=:assignment_operator\n");
89     else if (ch == '+')
90         printf("+=:arithmetic_operator\n");
91     else if (ch == '-')
92         printf("-=:arithmetic_operator\n");
93     else if (ch == '*')
94         printf("*=:arithmetic_operator\n");
95     else if (ch == '/')
96         printf("/=:arithmetic_operator\n");
97     else if (ch == '%')
98         printf("%=:arithmetic_operator\n");
99     else if (ch == '>')
100         printf(">=:relational_operator\n");
101     else if (ch == '<')
102         printf("<=:relational_operator\n");
103     else if (ch == '!')
104         printf("!=:logical_operator\n");
105 }
106
107 fclose(fp);
108 return 0;
109 }

```

Listing 5: A C program to Identification of Operators

2.5.3 Input & Output

Input:

```
1 if (a >= 10 && b != 5 ) a = a + 1 ;
```

Output:

```
>= : relational Operator  
&& : logical Operator  
!= : relational Operator  
= : assignment Operator  
+ : arithmetic Operator
```

2.5.4 Explanation of Code

Using simply if condition for character match.

2.6 Task 6: Identification of Punctuation and Special Symbols

2.6.1 Task Description

A program to Identification of Punctuation and Special Symbols.

2.6.2 Source Code

```
1  #include <stdio.h>
2  #include <stdlib.h>
3  #include <ctype.h>
4  #include <string.h>
5
6  int main()
7  {
8
9      FILE *fp = NULL;
10     char ch;
11     char new_word[40];
12     int index = 0;
13
14     fp = fopen("task2.txt", "r");
15     if (fp == NULL)
16     {
17         printf("file not found");
18         exit(1);
19     }
20
21     while ((ch = fgetc(fp)) != EOF)
22     {
23         if (ispunct(ch))
24         {
25             printf("%c: Punctuation Symbol\n", ch);
26         }
27         else if (!isdigit(ch) && !isalpha(ch) && !
                isspace(ch))
28         {
29             printf("%c: special Symbol\n", ch);
30         }
31     }
```

```

32
33     fclose(fp);
34
35     return 0;
36 }

```

Listing 6: A C program to Identification of Punctuation and Special Symbols

2.6.3 Input & Output

Input:

```
1 if(a%2==0) printf("%d is even", a);
```

Output:

```

( : Punctuation Symbol
) : Punctuation Symbol
{ : Punctuation Symbol
= : Punctuation Symbol
. : Punctuation Symbol
; : Punctuation Symbol
; : Punctuation Symbol
} : Punctuation Symbol

```

2.6.4 Explanation of Code

Using buiidin function to Identification of Punctuation and Special Symbols.

2.7 Task 7: Identification of Single-line and Multi-line Comments

2.7.1 Task Description

A program that detects and prints single-line comments (//...) and multi line comments (/...*/). Handle multiple comments in the same file and report unterminated comments as lexical errors.*

2.7.2 Source Code

```
1  #include <stdio.h>
2  #include <stdlib.h>
3  #include <ctype.h>
4  #include <string.h>
5
6  int main()
7  {
8
9      FILE *fp = NULL;
10     char ch, next_ch;
11
12     fp = fopen("task7.txt", "r");
13     if (fp == NULL)
14     {
15         printf("file not found");
16         exit(1);
17     }
18
19     while ((ch = fgetc(fp)) != EOF)
20     {
21         if (ch == '/')
22         {
23             next_ch = fgetc(fp);
24             if (next_ch == '/')
25             {
26                 printf("Single-line comments\n");
27                 while ((ch = fgetc(fp)) != EOF && ch
28                     != '\n')
29                     ;

```



```

30     else if (next_ch == '*')
31     {
32         int flag = 0;
33         while ((ch = fgetc(fp)) != EOF)
34         {
35             next_ch = fgetc(fp);
36             // printf("%c\n",ch);
37             // printf("%c\n", next_ch);
38             if (ch == '*' && next_ch == '/')
39             {
40                 flag = 1;
41                 break;
42             }
43             ungetc(next_ch, fp);
44         }
45
46         if (flag == 1)
47         {
48             printf("Multi-line comments\n");
49         }
50         else
51         {
52             printf("Unterminated Multi-line
53                     comments, Lexical error\n");
54         }
55     }
56 }
57
58 fclose(fp);
59
60 return 0;
61 }

```

Listing 7: A C program to Identification of Single-line and Multi-line Comments

2.7.3 Input & Output

Input:

```
1 // this is comment;  
2 /* multi line comment  
3 */  
4 /* this is
```

Output:

```
Single-line comments  
Multi-line comments  
Unterminated Multi-line comments, Lexical error
```

2.7.4 Explanation of Code

Using a flag variable and loop for comment condition .

2.8 Task 8: Identification of String Literals and Character Constants

2.8.1 Task Description

A program to detect valid and invalid string literals and character constants.

2.8.2 Source Code

```
1  #include <stdio.h>
2  #include <stdlib.h>
3  #include <string.h>
4
5  void check_string(const char *word)
6  {
7      int len = strlen(word);
8
9      if (len >= 2 && word[0] == '"' && word[len-1] == '
10         ')
11         printf("%s: Valid String Literal\n", word);
12     else if (len >= 2 && word[0] == '"' && word[len-1]
13         != '"')
14         printf("%s: Invalid String Literal\n", word);
15     else if (len >= 1 && word[0] == '"' && word[len-1]
16         != '"')
17         printf("%s: Unterminated String Literal\n",
18             word);
19     else if (len >= 2 && word[0] == '\\' && word[len
20         -1] == '\\')
21         printf("%s: Multiple Character Constant(
22             Invalid)\n", word);
23     else if (len == 3 && word[0] == '\\' && word[len-1]
24         == '\\')
25     {
26         printf("%s: valid Character Constant\n", word
27             );
28     }
29     else if (len == 2 && word[0] == '\\' && word[len-1]
30         == '\\')
31     {
```

```

23         printf("%s: Empty Character Constant(Invalid)
        \n", word);
24     }
25     else{
26         printf("%s: Unterminated Character Constant(
        Invalid)\n", word);
27     }
28
29 }
30
31
32 int main()
33 {
34     FILE *fp = fopen("task8.txt", "r");
35     if (!fp) {
36         printf("file not found");
37         return 1;
38     }
39
40     char ch;
41     char buffer[30];
42     int i;
43
44     while ((ch = fgetc(fp)) != EOF)
45     {
46         if (ch == '"')
47         {
48             i = 0;
49             buffer[i++] = ch;
50
51             while ((ch = fgetc(fp)) != EOF && ch != '\
                n')
52             {
53                 buffer[i++] = ch;
54                 if (ch == '"')
55                     break;
56             }
57
58             buffer[i] = '\0';
59             printf("@%s@\n", buffer);
60             check_string(buffer);

```

```

61         }
62     }
63
64     fclose(fp);
65     return 0;
66 }

```

Listing 8: A C program to Identification of String Literals and Character Constants

2.8.3 Input & Output

Input:

```

1  "a"
2  "hi"
3  "hello
4  'a'
5  ''
6  'ab'
7  'a

```

Output:

```

"a : Valid String Literal
"hi : Valid String Literal
"hello : Unterminated String Literal
'a' : valid Character Constant
''
' : Multiple Character Constant(Invalid)
' : Unterminated Character Constant(Invalid)
'a : Unterminated Character Constant(Invalid)

```

2.8.4 Explanation of Code

Word are seperated then first and last character to detect valid and invalid string literals and character constants

2.9 Task 9: Full Lexical Token Classification and Token Stream Generation

2.9.1 Task Description

A lexical analyzer that scans a file and classifies each lexeme

2.9.2 Source Code

```
1  #include <stdio.h>
2  #include <stdlib.h>
3  #include <ctype.h>
4  #include <string.h>
5
6  char *keywords[] = {
7      "auto", "break", "case", "char", "const", "continue", "
8      default", "do", "double",
9      "else", "enum", "extern", "float", "for", "goto", "if", "
10     inline", "int", "long",
11     "register", "restrict", "return", "short", "signed", "
12     sizeof", "static",
13     "struct", "switch", "typedef", "union", "unsigned", "
14     void", "volatile", "while"
15 };
16
17 int keyword_count = 34;
18
19 int is_keyword(const char *word) {
20     for (int i = 0; i < keyword_count; i++) {
21         if (strcmp(word, keywords[i]) == 0)
22             return 1;
23     }
24     return 0;
25 }
26
27 int is_identifier(const char *word) {
28     if (!isalpha(word[0]) && word[0] != '_')
29         return 0;
30
31     for (int i = 1; word[i] != '\0'; i++)
32         if (!isalnum(word[i]) && word[i] != '_')
33             return 0;
```

```

29
30     return !is_keyword(word);
31 }
32
33 int is_number(const char *word) {
34     int dots = 0;
35     for (int i = 0; word[i] != '\0'; i++) {
36         if (word[i] == '.') dots++;
37         else if (!isdigit(word[i])) return 0;
38     }
39     return dots <= 1;
40 }
41
42 int main() {
43     FILE *fp = fopen("task9.txt", "r");
44     if (!fp) {
45         printf("File not found!\n");
46         return 1;
47     }
48
49     char ch;
50
51     while ((ch = fgetc(fp)) != EOF) {
52
53         /* -----
54            IDENTIFIERS & KEYWORDS
55            ----- */
56         if (isalpha(ch) || ch == '_') {
57             char word[100];
58             int i = 0;
59             word[i++] = ch;
60
61             while ((ch = fgetc(fp)) != EOF && (isalnum
62                (ch) || ch == '_'))
63                 word[i++] = ch;
64
65             word[i] = '\0';
66             ungetc(ch, fp);
67
68             if (is_keyword(word))
69                 printf("%s: Keyword\n", word);

```

```

69         else
70             printf("%s: Identifier\n", word);
71
72         continue;
73     }
74
75     /* -----
76        NUMBERS (INTEGER / FLOAT)
77     ----- */
78     if (isdigit(ch)) {
79         char num[100];
80         int i = 0;
81         num[i++] = ch;
82
83         while ((ch = fgetc(fp)) != EOF && (isdigit
84             (ch) || ch == '.'))
85             num[i++] = ch;
86
87         num[i] = '\0';
88         ungetc(ch, fp);
89
90         if (is_number(num)) {
91             if (strchr(num, '.'))
92                 printf("%s: Float Constant\n",
93                     num);
94             else
95                 printf("%s: Integer Constant\n",
96                     num);
97         } else {
98             printf("%s: Invalid Number\n", num);
99         }
100         continue;
101     }
102
103     /* -----
104        STRING LITERALS
105     ----- */
106     if (ch == '"') {
107         char buffer[200];
108         int i = 0;
109         buffer[i++] = ch;

```



```

107
108     while ((ch = fgetc(fp)) != EOF && ch != '"
109         ' ) {
110         if (ch == '\\n') break;
111         buffer[i++] = ch;
112
113     buffer[i++] = '"';
114     buffer[i] = '\\0';
115
116     if (ch == '"')
117         printf("%s: _String_Literal\\n", buffer
118             );
119     else
120         printf("%s: _Unterminated_String_
121             Literal\\n", buffer);
122
123     continue;
124 }
125
126 /* -----
127     CHARACTER CONSTANTS
128     ----- */
129 if (ch == '\\') {
130     char buf[10];
131     int i = 0;
132     buf[i++] = ch;
133     buf[i++] = fgetc(fp);
134     buf[i++] = fgetc(fp);
135     buf[i] = '\\0';
136
137     if (buf[2] == '\\')
138         printf("%s: _Character_Constant\\n",
139             buf);
140     else
141         printf("%s: _Invalid_Character_
142             Constant\\n", buf);
143
144     continue;
145 }

```

```

143      /* -----
144      COMMENTS
145      ----- */
146      if (ch == '/') {
147          char next = fgetc(fp);
148
149          if (next == '/') {
150              printf("//: Single-line Comment\n");
151              while ((ch = fgetc(fp)) != EOF && ch
152                  != '\n');
153              continue;
154          }
155
156          else if (next == '*') {
157              printf("/**/: Multi-line Comment\n")
158              ;
159              char prev = 0;
160              while ((ch = fgetc(fp)) != EOF) {
161                  if (prev == '*' && ch == '/')
162                      break;
163                  prev = ch;
164              }
165              continue;
166          }
167
168          ungetc(next, fp);
169      }
170
171      /* -----
172      OPERATORS
173      ----- */
174      char next = fgetc(fp);
175      if (next != EOF) {
176          char op[3] = {ch, next, '\0'};
177
178          if (!strcmp(op, "==") || !strcmp(op, "!=")
179              ||
180              !strcmp(op, ">=") || !strcmp(op, "<=")
181              )
182          { printf("%s: Relational Operator\n", op)
183              ; continue; }

```

```

178
179         if (!strcmp(op, "++") || !strcmp(op, "--"))
180         { printf("%s: Arithmetic Operator\n", op)
181           ; continue; }
182
183         if (!strcmp(op, "&&") || !strcmp(op, "||"))
184         { printf("%s: Logical Operator\n", op);
185           continue; }
186
187         if (!strcmp(op, "+=") || !strcmp(op, "-=")
188             ||
189             !strcmp(op, "*=") || !strcmp(op, "/=")
190         )
191         { printf("%s: Assignment Operator\n", op)
192           ; continue; }
193
194         ungetc(next, fp);
195     }
196
197     /* Single-character operators */
198     if (strchr("=+-*/%><!&|", ch)) {
199         printf("%c: Operator\n", ch);
200         continue;
201     }
202
203     /* -----
204        SPECIAL SYMBOLS & PUNCTUATIONS
205        ----- */
206     if (ispunct(ch)) {
207         printf("%c: Special Symbol\n", ch);
208     }
209 }
210
211 fclose(fp);
212 return 0;
213 }

```

Listing 9: A C program for Full Lexical Token Classification and Token Stream Generation

2.9.3 Input & Output

Input:

```
1 int x = 10 ;  
2 float y = 2.5 ;  
3 printf("Hello") ;
```

Output:

```
int : Keyword  
x : Identifier  
= : Operator  
10 : Integer Constant  
; : Special Symbol  
float : Keyword  
y : Identifier  
= : Operator  
2.5 : Float Constant  
; : Special Symbol  
printf : Identifier  
( : Special Symbol  
"Hello" : String Literal  
) : Special Symbol  
; : Special Symbol
```

2.9.4 Explanation of Code

Combine previous problem code methods.

2.10 Task 10: Detection of Lexical Errors

2.10.1 Task Description

A program that detects and reports common lexical errors:

- *Invalid identifiers:* `2sum`, `a-bc`, `@total`
- *Unterminated strings:* `"hello`
- *Unclosed comments:* `/* comment`
- *Invalid characters:* `@`, `(` (in identifiers)

2.10.2 Source Code

```
1  #include <stdio.h>
2  #include <ctype.h>
3  #include <string.h>
4
5  int is_keyword(const char *word) {
6      char *keywords[] = {
7          "int", "float", "char", "double", "long", "short", "
8          void", "return",
9          "if", "else", "while", "for", "break", "continue", "
10         struct", "union"
11     };
12     int n = sizeof(keywords) / sizeof(keywords[0]);
13     for (int i = 0; i < n; i++) {
14         if (strcmp(word, keywords[i]) == 0)
15             return 1;
16     }
17     return 0;
18 }
19
20 int is_valid_identifier(const char *word) {
21     if (!isalpha(word[0]) && word[0] != '_')
22         return 0;
23
24     for (int i = 1; word[i] != '\0'; i++)
25         if (!isalnum(word[i]) && word[i] != '_')
26             return 0;
27
28     return 1;
29 }
30
31 int main() {
```

```

31 FILE *fp = fopen("task10.txt", "r");
32 if (!fp) {
33     printf("Cannot open file.\n");
34     return 1;
35 }
36
37 char ch;
38
39 while ((ch = fgetc(fp)) != EOF) {
40
41     /* =====
42        INVALID IDENTIFIERS (start with digit,
43        contain symbols)
44        ===== */
45     if (isalnum(ch) || ch == '_' || ch == '@' ||
46         ch == '$' || ch == '#') {
47
48         char word[100];
49         int i = 0;
50         word[i++] = ch;
51
52         while ((ch = fgetc(fp)) != EOF &&
53             (isalnum(ch) || ch == '_' || ch ==
54              '@' || ch == '$' || ch == '#'))
55         {
56             word[i++] = ch;
57         }
58
59         word[i] = '\0';
60         ungetc(ch, fp);
61
62         int invalid_char = 0;
63         for (int j = 0; word[j]; j++) {
64             if (word[j] == '@' || word[j] == '$'
65                 || word[j] == '#')
66                 invalid_char = 1;
67         }
68
69         if (isdigit(word[0]) || invalid_char || !
70             is_valid_identifier(word)) {

```

```

66         printf("Error: Invalid identifier '%s\n", word);
67     }
68
69     continue;
70 }
71
72 /* =====
73    UNTERMINATED STRING
74 ===== */
75 if (ch == '"') {
76     char buffer[200];
77     int i = 0;
78     buffer[i++] = ch;
79     int closed = 0;
80
81     while ((ch = fgetc(fp)) != EOF) {
82         if (ch == '"') {
83             closed = 1;
84             buffer[i++] = ch;
85             break;
86         }
87         if (ch == '\n')
88             break;
89
90         buffer[i++] = ch;
91     }
92
93     buffer[i] = '\0';
94
95     if (!closed)
96         printf("Error: Unterminated string literal '%s\n", buffer);
97
98     continue;
99 }
100
101 /* =====
102    UNCLOSED COMMENTS /* comment ...
103 ===== */
104 if (ch == '/') {

```

```

105         char next = fgetc(fp);
106
107         if (next == '*') {
108             int closed = 0;
109             char prev = 0;
110
111             while ((ch = fgetc(fp)) != EOF) {
112                 if (prev == '*' && ch == '/') {
113                     closed = 1;
114                     break;
115                 }
116                 prev = ch;
117             }
118
119             if (!closed)
120                 printf("Error: _Unclosed _comment _
121                        _/'_\\n");
122
123             continue;
124         }
125
126         ungetc(next, fp);
127
128         /* =====
129         INVALID STANDALONE CHARACTERS
130         ===== */
131         if (ch == '@' || ch == '$' || ch == '#') {
132             printf("Error: _Invalid _character _'%c'_\\n",
133                    ch);
134         }
135
136         fclose(fp);
137         return 0;
138     }

```

Listing 10: A C program to Detection of Lexical Errors

2.10.3 Input & Output

Input:

```
1 int 2sum = 10 ;  
2 char *s = "Hello" ;  
3 float x@ = 2.5;
```

Output:

```
Error: Invalid identifier '2sum'  
Error: Invalid identifier '10'  
Error: Invalid identifier 'x@'  
Error: Invalid identifier '2'  
Error: Invalid identifier '5'
```

2.10.4 Explanation of Code

Using previous code methods.

3 Conclusion

Lexical analysis is an essential phase of the compiler where the source code is scanned and broken into meaningful units called tokens. It acts as a bridge between raw program text and the syntax analyzer, ensuring that the input program is structured, valid, and easy to process in later stages. In this lab, I implemented basic lexical analysis tasks such as identifying keywords, operators, special symbols, and comments. I also detected single-line and multi-line comments and handled error cases like unterminated comment blocks. Through these tasks, I gained practical understanding of how a lexical analyzer processes characters, builds tokens, and performs error detection before syntax analysis begins.