Dhaka International University
Department of Computer Science and Engineering
Dhaka, Bangladesh
10th Semester

Lab Report: 02

Computer Networking Lab (CSE-402)
Submit Date: November 10, 2025

Submitted by:	**Submitted to:**
Tohidul Islam Rupok ID: CS-D-77-22-120014 Roll: 25 Batch: D-77 Department of Computer Science and Engineering (CSE) Email: tohidrupok@students.diu.ac	Mazharul Hasan Lecturer, Department of Computer Science and Engineering (CSE) Mobile: +8801631981121 Email: mazharulhasan.cse@diu.ac

Knowledge is Power | Dhaka International University

Report Number: 02

Report Title: Checking Identifier, Number, and Keywords using Flex

Objective

The objective of this experiment is to design a lexical analyzer using Flex that can read an input program and identify identifiers, numbers, and keywords.

The program will classify each token from the input text and display its category as output.

Algorithm

- **Step 1:** Start the program.
- **Step 2:** Define patterns for identifiers, numbers, and keywords using regular expressions.
- **Step 3:** Create a list of keywords (int, float, if, else…).
- **Step 4:** Read input text from the user.
- **Step 5:**

  → If a token matches the number pattern, print "NUMBER".

  → If a token matches the identifier pattern, check whether it's a keyword:

  → If yes → print "KEYWORD"

  → Otherwise → print "IDENTIFIER"

- **Step 6:** Ignore spaces, tabs, and newlines.
- **Step 7:** For any unmatched characters, print "OTHER".
- **Step 8:** Stop the program.

Code Implementation

```
%option noyywrap

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

```
// List of keywords //
char *keywords[] = {
    "int", "float", "if", "else", "while", "for", "return", "break", "char", "double", "void"
};

// Function to check  keyword //
int isKeyword(char *word) {
    for (int i = 0; i < sizeof(keywords)/sizeof(keywords[0]); i++) {
        if (strcmp(word, keywords[i]) == 0)
            return 1;
    }
    return 0;
}
%}

// Regular expressions //

digit       [0-9]
letter      [A-Za-z_]
id          {letter}({letter}|{digit})*
number      {digit}+(\.{digit}+)?(E[+-]?{digit}+)?
delim       [ \t\n]+

%%

{number} {
    printf("NUMBER\t\t: %s\n", yytext);
}

{id} {
    if (isKeyword(yytext))
        printf("KEYWORD\t\t: %s\n", yytext);
    else
        printf("IDENTIFIER\t: %s\n", yytext);
}

{delim} ;

. {
    printf("OTHER\t\t: %s\n", yytext);
}

%%

int main() {
    printf("Enter your input:\n");
    yylex();
    return 0;
}
```

Input/ Output

```
PS C:\Users\HP\Downloads\17-CSE4102-main\17-CSE4102-main\Lab2> flex rupok.l
PS C:\Users\HP\Downloads\17-CSE4102-main\17-CSE4102-main\Lab2> gcc lex.yy.c -o rupok.exe
PS C:\Users\HP\Downloads\17-CSE4102-main\17-CSE4102-main\Lab2> ./rupok.exe
Enter your input:
int x = 10;
KEYWORD         : int
IDENTIFIER      : x
OTHER           : =
NUMBER          : 10
OTHER           : ;
123.45E-2
NUMBER          : 123.45E-2
aaa
IDENTIFIER      : aaa
25
NUMBER          : 25
```

Analysis & Conclusion

In this lab, I successfully built a lexical analyzer using Flex that recognizes identifiers, numbers, and keywords from a given input program.
The program uses regular expressions to define token patterns and a C function to compare tokens with a predefined keyword list.

Challenges faced:

- Understanding how to combine Flex regular expressions with C code.
- Handling floating-point and scientific number formats.
- Managing whitespace and special symbols properly.

Conclusion:
The program correctly identifies and classifies input tokens into their respective categories. This experiment demonstrates how lexical analysis is used in compiler design to break source code into meaningful tokens for further processing.