Objective

To develop a basic lexical analyzer using Flex that can recognize and categorize different components (lexemes), such as keywords, identifiers, numbers, and symbols from a line of code.

Required Tools & Setup

- A code editor (VS Code)
- Linux environment or Windows Subsystem for Linux (WSL)
- GCC (GNU Compiler Collection)
- Latest version of Flex (Fast Lexical Analyzer)

Theory & Approach

In this lab, we implemented a simple lexical analyzer using Flex, a powerful tool for generating scanners. Our analyzer accepts an input string and breaks it into categorized tokens.

Structure of a Flex Program:

1. **Definition Section (`%{ %}`):**
   Here, we include necessary C headers and global declarations.
   Example:
   %{ #include<stdio.h> %}

2. **Rules Section (`%% ... %%`):**
   This contains pattern-action pairs. The left side defines a regular expression; the right side defines the action (what to do if the pattern matches).
   Key patterns:

   a. **Keywords:** int | float | if | else | for | do | while | return | break | char | const | continue | default | switch | void

   b. **Identifiers:** [a-z A-Z_] [a-z A-Z 0-9_]*

   c. **Numbers:** [0-9]+(\.[0-9]+)?([e][+-]?[0-9]+)?

   d. **Others:** Any other symbol matched using `.`

Code Implementation

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

%%
int|float|if|else|for|do|while|return|break    {printf("'%s' is a keyword\n", yytext);}
[a-zA-Z_][a-zA-Z0-9_]*    {printf("'%s' is an Identifier\n", yytext);}
[0-9]+(\.[0-9]+)?([e][+-]?[0-9]+)?    {printf("'%s' is a Number\n", yytext);}
.           { printf("Others symbol\n"); }
\n          { return 0;}
%%

int yywrap()
{
   return 1;
}

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

How to Compile and Run

Open a terminal and run:

1. Run file_name.l; it will generate lex.yy.c
2. Run gcc lex.yy.c; it will generate a.out
3. Run a.out

Provide an input line like:

float number is 3.3*10e8

Output

```
soumik@DESKTOP-KH1FLQO:/mnt/d/Classroom/10th Semester/Compiler Design/Bison/Flex$ flex flex.l
soumik@DESKTOP-KH1FLQO:/mnt/d/Classroom/10th Semester/Compiler Design/Bison/Flex$ gcc lex.yy.c
soumik@DESKTOP-KH1FLQO:/mnt/d/Classroom/10th Semester/Compiler Design/Bison/Flex$ ./a.out
Enter a string :
float number is 3.3*10e8
'float' is a keyword
Others symbol
'number' is an Identifier
Others symbol
'is' is an Identifier
Others symbol
'3.3' is a Number
Others symbol
'10e8' is a Number
```

Conclusion

This lab successfully introduced the basics of lexical analysis using Flex. We explored how to define token patterns using regular expressions and how to process inputs to classify them. Despite facing challenges with environment setup and syntax precision, the practical experience significantly enhanced our understanding of how compilers begin the translation process through lexical analysis.