BANGLADESH UNIVERSITY OF PROFESSIONALS

Department of Computer Science and Engineering
Faculty of Science and Technology

Compiler Laboratory

CSE-3104

SUBMITTED TO:

Rumana Yasmin
Lecturer
Department of Computer Science and Engineering
Bangladesh University of Professionals

Tanjim Taharat Aurpa
Lecturer
Department of Data Science and Engineering
Gazipur Digital University

SUBMITTED BY:

Towfiq Omar Rakin
ID 23524202131
BCSE-2023
Section A

DATE:

24th September 2025

Task 01

Write a flex program to recognize-

- **Numbers** (123, 345 etc)
- **Words** (hello, world etc)

Code

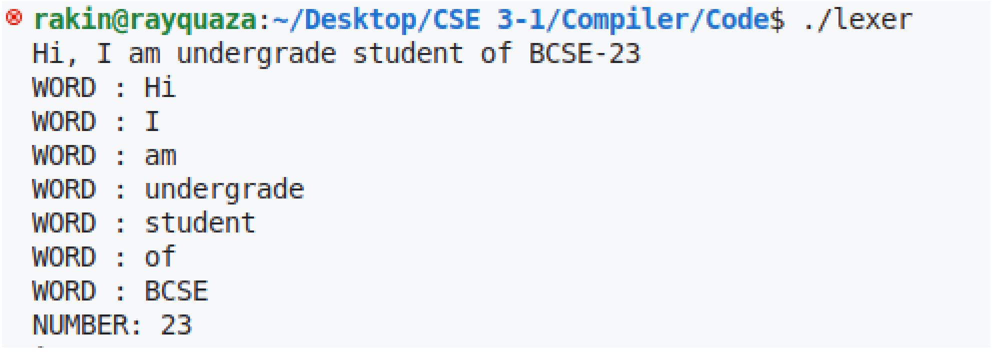
```
%{
#include <stdio.h>
%}

%%
[0-9]+       { printf("NUMBER: %s\n", yytext); }
[a-zA-Z]+    { printf("WORD : %s\n", yytext); }
.|\n         { /* ignore other characters */ }
%%

int yywrap(void) {
   return 1;
}

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

Output



```
rakin@rayquaza:~/Desktop/CSE 3-1/Compiler/Code$ ./lexer
Hi, I am undergrade student of BCSE-23
WORD : Hi
WORD : I
WORD : am
WORD : undergrade
WORD : student
WORD : of
WORD : BCSE
NUMBER: 23
```

Task 02

Write a Flex program to recognize arithmetic operators: **+, -, *,** /

Code

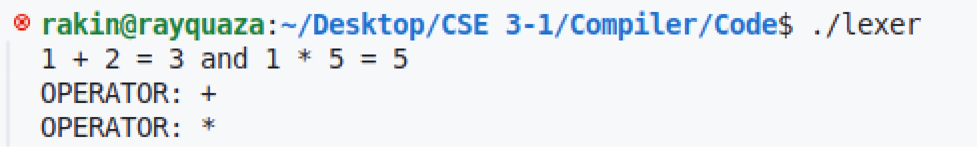
```
%{
#include <stdio.h>
%}

%%
[+\-*/]        { printf("OPERATOR: %s\n", yytext); }
.|\n              { /* ignore other characters */ }
%%

int yywrap(void) {
   return 1;
}

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

Output



```
⊗ rakin@rayquaza:~/Desktop/CSE 3-1/Compiler/Code$ ./lexer
  1 + 2 = 3 and 1 * 5 = 5
  OPERATOR: +
  OPERATOR: *
```

Task 03

Write a Flex program to count the number of vowels and consonants in the given input string.

Code

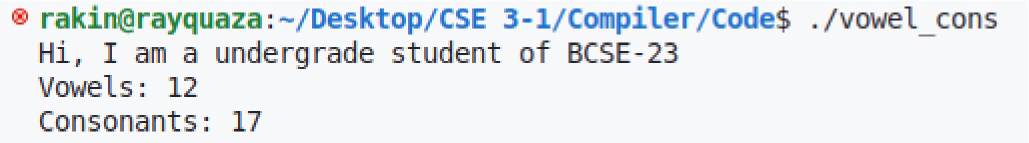
```
%{
#include <stdio.h>
int vowel_count = 0;
int consonant_count = 0;
%}

%%
[AEIOUaeiou]    { vowel_count++; }
[A-Za-z]        { consonant_count++; }
.               { /* ignore other characters */ }
\n              { printf("Vowels: %d\nConsonants: %d\n", vowel_count,
consonant_count);}
%%

int yywrap(void) { return 1; }

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

Output



```
rakin@rayquaza:~/Desktop/CSE 3-1/Compiler/Code$ ./vowel_cons
Hi, I am a undergrade student of BCSE-23
Vowels: 12
Consonants: 17
```

Task 04

Write a Flex program to identify valid floating-point numbers (e.g., 12.34, 0.56, 45.0).

Code

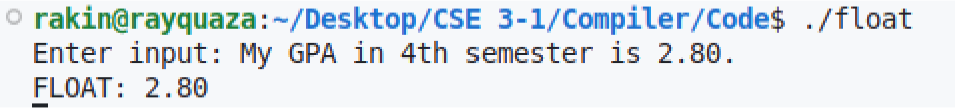
```
%{
#include <stdio.h>
%}

%%
[0-9]+\.[0-9]+    { printf("FLOAT: %s\n", yytext); }
.|\n              { /* ignore other characters */ }
%%
```

```
int yywrap(void) { return 1; }

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

Output



```
rakin@rayquaza:~/Desktop/CSE 3-1/Compiler/Code$ ./float
Enter input: My GPA in 4th semester is 2.80.
FLOAT: 2.80
```

Questions

1. **What is Lex tool used for?**

   Ans: Lex is a tool for generating lexical analyzers. It reads an input stream of characters and converts it into tokens, which are meaningful symbols used by compilers to understand the input program.

2. **List the different sections available in Lex code?**

   Ans: Lex code is divided into three sections separated by "%%":

   i. Declarations: definitions and code included in %{ %} for global use.

   ii. Rules: patterns (regular expressions) and actions (C code) for matching tokens.

   iii. Auxiliary routines: additional C functions used in the rules section.

3. **What is the input for Lex Compiler?**
   Ans: The input for the Lex compiler is a source program written in the Lex language, which describes the lexical analyzer using token patterns and actions. This source file typically has a .l or .lex extension.

4. **What is the output of a Lex compiler?**
   Ans: The output of the Lex compiler is a C source file named lex.yy.c. This C file contains the generated lexical analyzer code. This file is then compiled by a C compiler to produce an executable lexical analyzer that can process input streams and generate tokens.

5. **Why do we need a lexical analyzer?**
   Ans: A lexical analyzer is needed to break down the input stream of characters from source code into meaningful units called tokens. This simplifies the syntax analysis phase of compilation by categorizing input into manageable input into manageable.

6. **What does a lexical analyzer do when prefixes of the input string match more than one pattern?**
   Ans: When prefixes of the input string match more than one pattern, the lexical analyzer selects the longest matching pattern. If multiple patterns match the same longest prefix, it chooses the one that appears first in the Lex specification. This ensures that the token recognized is the one that consumes the most input characters, resolving ambiguities in favor of the longest match and priority order in the rules.