



**BANGLADESH UNIVERSITY OF
PROFESSIONALS**
FACULTY OF SCIENCE & TECHNOLOGY
DEPT. OF COMPUTER SCIENCE & ENGINEERING (CSE)



COURSE NAME: Compiler Laboratory

CODE: CSE-3104

TITLE : Introduction to Lex, YACC and Flex/Bison Installation

SUBMITTED TO:

Rumana Yasmin
Lecturer
Department of CSE
FST, BUP

Sayma Alam Suha
Lecturer
Department of CSE
FST, BUP

SUBMITTED BY:

NAME : Zuairia Kabir
ROLL : 2252421013
SECTION : A
LEVEL/TERM : 3rd

SEMESTER: 1st

DATE OF SUBMISSION: October 26, 2024

Experiment number:01

Title: Basic Lex program

Introduction:

Lexical analysis is an essential phase in the process of program compilation and interpretation. During this phase, the source code is scanned and broken down into fundamental units, called tokens, which represent keywords, operators, identifiers, and other symbols within the code. This stage is crucial, as it prepares the code for the next step, syntax analysis, where a parse tree is constructed based on the tokenized structure. In this lab, we will perform lexical analysis using Flex (Fast Lexical Analyzer Generator), a widely-used tool designed to create lexical analyzers for programs written in the lex language.

Additionally, we will incorporate YACC (Yet Another Compiler Compiler), a parser generator that works in conjunction with Flex to produce syntax parsers, and the GNU GCC compiler to compile our code. Together, these tools form a powerful pipeline for analyzing and compiling source code. Flex will tokenize the code, YACC will create a parse file to structure the token sequences, and GNU GCC will compile the final program. This lab will be carried out via Command Prompt, where we will write lex language code, execute tokenization, and compile the code. By the end, we'll gain insight into the roles of lexical analysis and syntax parsing in program compilation.

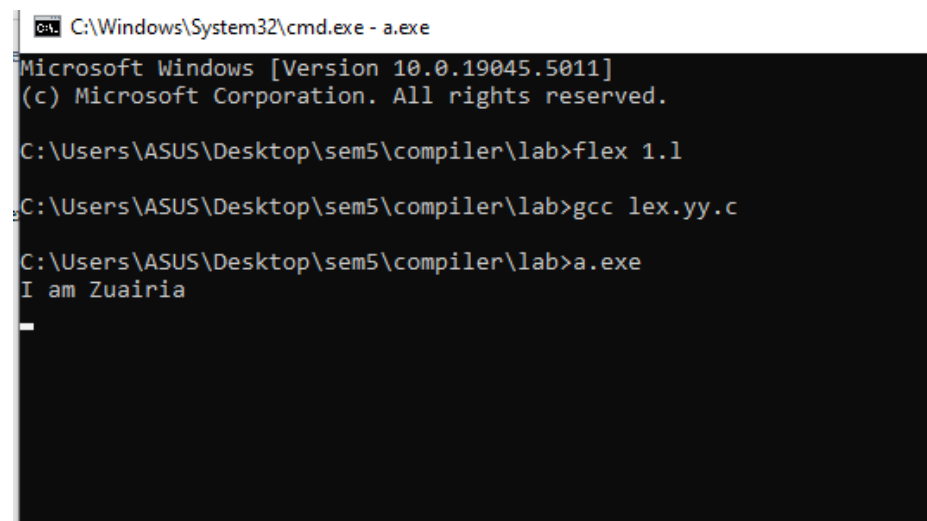
Objective:

- 1.To learn the basics of lexical analysis
- 2.To familiarize with the syntax and structure of Lex/Flex programs
- 3.To learn how to define and use regular expressions within Lex/Flex

1.Task/Code:

```
% {  
#include<stdio.h>  
% }  
%%  
printf("I am Zuairia\n");  
%%  
int yywrap(void){  
return 1;  
}  
int main(){  
yylex();  
return 0;  
}
```

1.Input/Output:

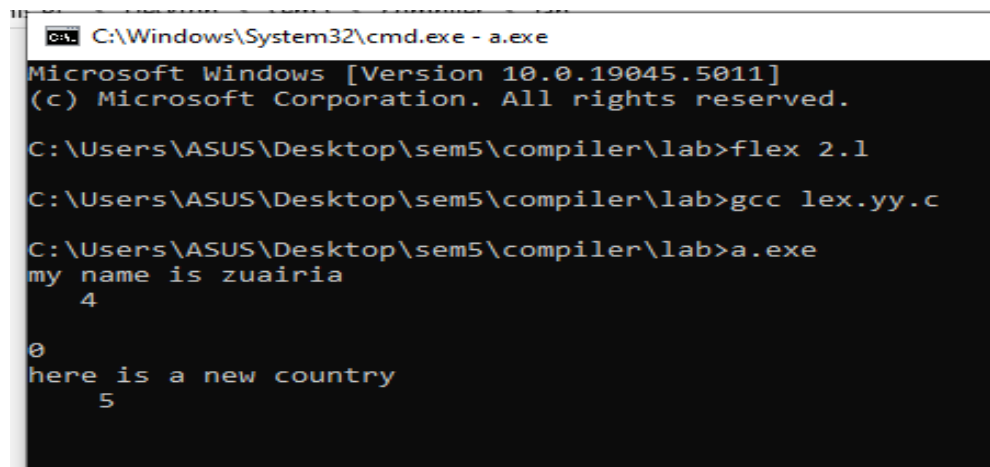


```
C:\Windows\System32\cmd.exe - a.exe  
Microsoft Windows [Version 10.0.19045.5011]  
(c) Microsoft Corporation. All rights reserved.  
  
C:\Users\ASUS\Desktop\sem5\compiler\lab>flex 1.1  
  
C:\Users\ASUS\Desktop\sem5\compiler\lab>gcc lex.yy.c  
  
C:\Users\ASUS\Desktop\sem5\compiler\lab>a.exe  
I am Zuairia  
-
```

2.Task/Code:

```
% {  
#include<stdio.h>  
#include<string.h>  
int cnt=0;  
% }  
%%  
[a-zA-Z0-9]* {cnt++;}  
"\n" {printf("%d\n",cnt); cnt=0;}  
%%  
int yywrap(void){ }  
int main()  
{  
yylex();  
return 0;  
}
```

2.Input/Output:

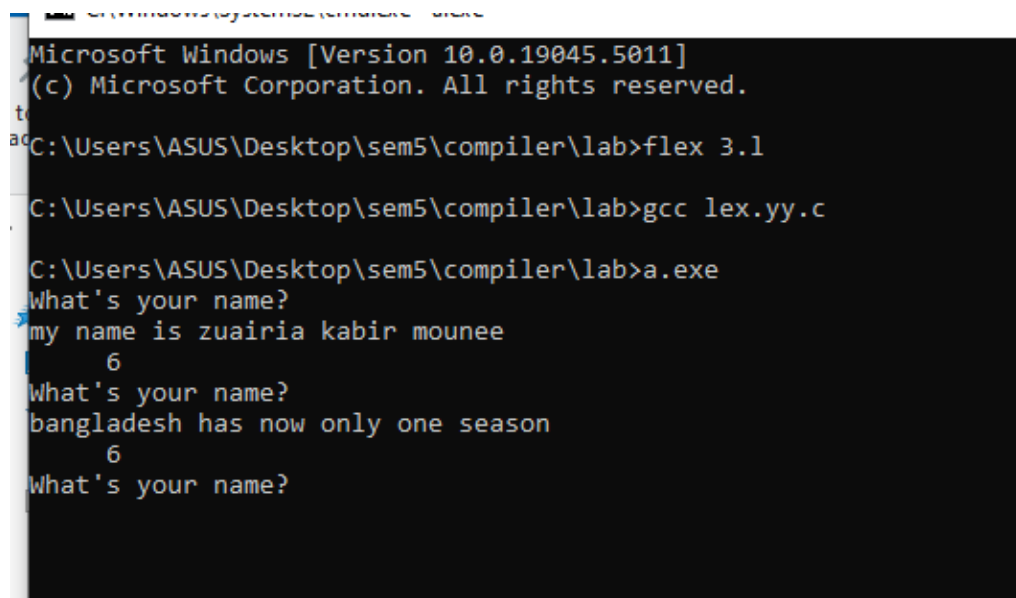


```
C:\Windows\System32\cmd.exe - a.exe  
Microsoft Windows [Version 10.0.19045.5011]  
(c) Microsoft Corporation. All rights reserved.  
  
C:\Users\ASUS\Desktop\sem5\compiler\lab>flex 2.1  
C:\Users\ASUS\Desktop\sem5\compiler\lab>gcc lex.yy.c  
C:\Users\ASUS\Desktop\sem5\compiler\lab>a.exe  
my name is zuairia  
4  
  
0  
here is a new country  
5
```

3.Task/Code:

```
% {  
#include<stdio.h>  
#include<string.h>  
int cnt=0;  
% }  
%%  
[a-zA-Z0-9]* {cnt++;}  
"\n" {printf("%d\nWhat's your name?\n",cnt); cnt=0;}  
%%  
int yywrap(void){ }  
int main()  
{  
printf("What's your name?\n");  
yylex();  
return 0;  
}
```

3.Input/Output:



```
Microsoft Windows [Version 10.0.19045.5011]  
(c) Microsoft Corporation. All rights reserved.  
C:\Users\ASUS\Desktop\sem5\compiler\lab>flex 3.1  
C:\Users\ASUS\Desktop\sem5\compiler\lab>gcc lex.yy.c  
C:\Users\ASUS\Desktop\sem5\compiler\lab>a.exe  
What's your name?  
my name is zuairia kabilr mounes  
6  
What's your name?  
bangladesh has now only one season  
6  
What's your name?
```

Discussion:

This lab provided hands-on experience with Lex/Flex and introduced key concepts in lexical analysis, such as tokenization and regular expressions. By working through examples, we learned how Lex programs are structured and how to define rules to identify specific patterns, like words and numbers. By working through examples, we saw how lexical analyzers read text line-by-line, making it possible to count words or respond to user input. These basic skills are useful for working with text, building compilers, or creating programs that analyze language. Overall, Lex/Flex provides a good starting point for creating more complex text-processing tools in programming.