Experiment No:01

Experiment Name: Write a lex program to generate a string. Ex: Welcome To NUBTK

Objectives:

1. To understand the basic syntax and structure of a Flex program.
2. To learn how to write patterns for matching strings using regular expressions in Flex.
3. To implement a Flex scanner that can read and print any given input string.
4. To gain hands-on experience in compiling and running Flex programs.
5. To familiarize with the interaction between Flex and C language code for input/output operations.
6. To understand how lexical analyzers work in recognizing and processing input strings.

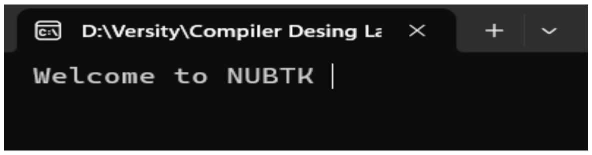
Tools:

• Flex Software
• Windows Terminal (CMD)

Source Code:

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

%%

%%
int main()
{
        printf(" Welcome To NUBTK");
        yylex();
        return 0;
}
```

Input &Output:



```
Welcome to NUBTK
```

Experiment No:02

Experiment Name: Write a lex program to Identity Uppercase & Lowercase. Ex: MahfujulKarim

---

Objectives:

1. To understand the difference between uppercase and lowercase characters in ASCII.
2. To write Flex patterns to detect uppercase and lowercase alphabets.
3. To create a Flex program that identifies and distinguishes uppercase letters from lowercase letters.
4. To practice using character classes and regular expressions in Flex.
5. To output the classification result for each character in the input string.

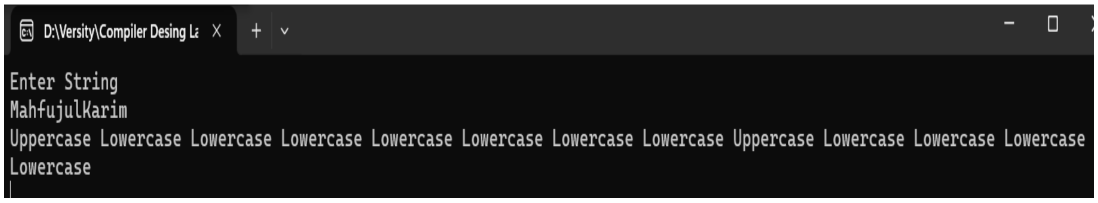
Tools:

• Flex Software
• Windows Terminal (CMD)

Source Code:

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

%%
[A-Z] {printf("Uppercase ");}
[a-z] {printf("Lowercase ");}
%%
int main()
{
        printf("Enter String\n");
        yylex();
        return 0;
}
```

Input &Output:



```
Enter String
MahfujulKarim
Uppercase Lowercase Lowercase Lowercase Lowercase Lowercase Lowercase Lowercase Uppercase Lowercase Lowercase Lowercase
Lowercase
```

Experiment No:03

Experiment Name: Write a lex program to Identity Vowel & Consonant. Ex: myfathermother

---

Objectives:

1. To understand how to use Flex to recognize specific character classes (vowels and consonants).
2. To write Flex patterns for vowels and consonants using regular expressions.
3. To develop a lexical analyzer that can differentiate vowels from consonants in input strings.
4. To print appropriate output based on the identification of vowels and consonants.
5. To practice working with character matching and conditional actions in Flex.

Tools:

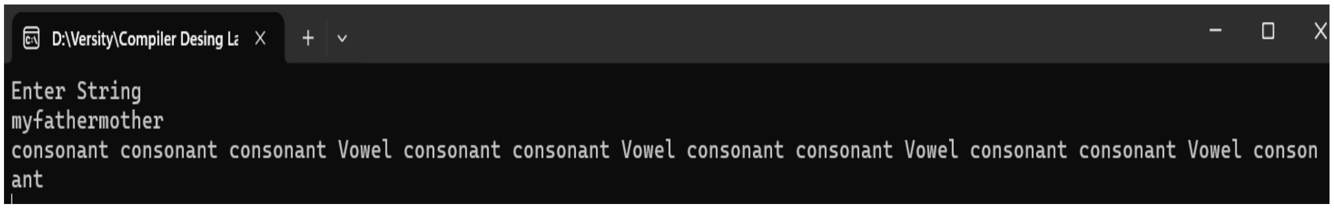
• Flex Software
• Windows Terminal (CMD)

Source Code:

```
%option noyywrap
%{
        #include<stdio.h>
        int v_c=0;
        int c_c=0;
%}

%%
[aeiou|AEIOU] {printf("Vowel ");}
[a-z|A-Z] {printf("consonant ");}
%%
```

```
int main()
{
        printf("Enter String\n");
        yylex();
        getch();
        return 0;
}
```

Input &Output:



```
Enter String
myfathermother
consonant consonant consonant Vowel consonant consonant Vowel consonant consonant Vowel consonant consonant Vowel conson
ant
```

Experiment No:04

Experiment Name: Write a lex program to count Vowel & Consonant. Ex:NorthernUniversityKhulna

---

Objectives:

1. To implement counters in Flex to keep track of vowels and consonants found in the input.
2. To reinforce the use of regular expressions for pattern matching in Flex.
3. To learn how to maintain and update variables within Flex actions.
4. To output the total count of vowels and consonants after processing the input string.
5. To develop skills in combining lexical analysis with simple computation logic.

Tools:

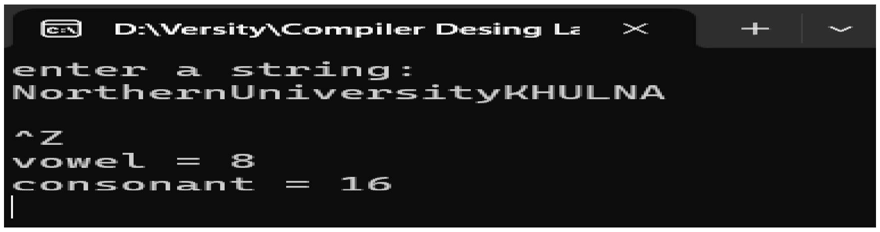
• Flex Software
• Windows Terminal (CMD)

Source Code:

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

%%
[AEIOU|aeiou]       { vowel_count++; }
[A-Z|a-z]         { consonant_count++; }

%%
```

```
int main()
{

   printf("enter a string:\n");
   yylex();
   printf("vowel = %d\n", vowel_count);
   printf("consonant = %d\n",
consonant_count);
   getch();
   return 0;
}
```

Input &Output:



```
enter a string:
NorthernUniversityKHULNA

^Z
vowel = 8
consonant = 16
```

Experiment No:05

Experiment Name: Write a lex program to identify Pattern (Digit/Numbers). Ex: if 3 123 abc

---

Objectives:

1. To learn how to identify digits and number patterns using regular expressions in Flex.
2. To differentiate between single digits and multi-digit numbers in input.
3. To write appropriate patterns for matching integers using Flex rules.
4. To understand how Flex processes numerical input and classifies it.
5. To output specific messages based on whether input is a digit or a number.

Tools:

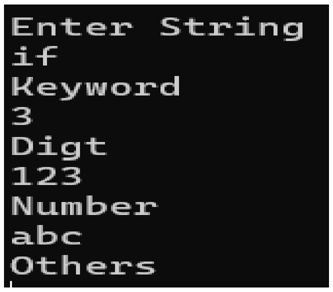
• Flex Software
• Windows Terminal (CMD)

Source Code:

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

%%
[0-9]  {printf("Digt ");}
[0-9]* {printf("Number");}
"if"|"else"|"while"|"do"|"switch"|"case"
{printf("Keyword");}
.* {printf("Others");}

%%
```

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

Input &Output:



```
Enter String
if
Keyword
3
Digt
123
Number
abc
Others
```

Experiment No:06

Experiment Name: Write a lex program to count Lines and Spaces from user input.

Ex: Hello World

This is Test

Objectives:

1. To implement a Flex program that counts the number of lines and spaces in a given input.
2. To learn how to detect newline and whitespace characters using regular expressions.
3. To maintain counters within Flex rules for tracking line breaks and spaces.
4. To reinforce the use of action blocks in Flex for computation.
5. To display the total number of lines and spaces after scanning the input.

Tools:

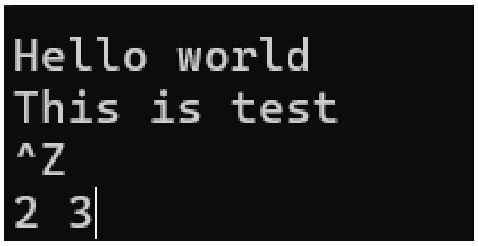
• Flex Software
• Windows Terminal (CMD)

Source Code:

```
%option noyywrap
%{
        #include<stdio.h>
        int line=0,other=0,space=0;
%}

%%
\n {line++;}
[ ] {space++;}
. {other++;}
%%
```

```
int main()
{
yylex();
printf("%d %d",line,space);
getch();
        return 0;
}
```

Input &Output:



```
Hello world
This is test
^Z
2 3
```

Experiment No:07

Experiment Name: Write a lex program to count positive integers, negative integers, positive floating-point numbers and negative floating-point numbers from user input. Ex: 123 -45 6.78 -0.9

Objectives:

1. To understand how to write regular expressions in Lex to recognize different types of numeric inputs.
2. To differentiate between positive and negative integers using pattern matching.
3. To identify positive and negative floating-point numbers by defining suitable Lex patterns.
4. To implement counters that keep track of each numeric category during input scanning.
5. To practice integrating Lex actions for counting and displaying results.

Tools:

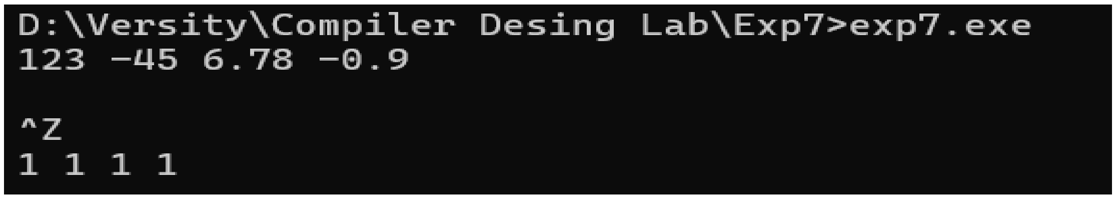
• Flex Software
• Windows Terminal (CMD)

Source Code:

```
%option noyywrap
%{
   #include <stdio.h>
   int p=0, n=0, pf=0, nf=0;
%}

%%

-[0-9]+            { n++; }
-[0-9]+\.[0-9]+    { nf++; }
[0-9]+\.[0-9]+     { pf++; }
[0-9]+             { p++; }

%%
```

```
int main()
{
   yylex();
   printf("%d %d %d %d\n", p, n, pf, nf);
   return 0;
}
```

Input &Output:



```
D:\Versity\Compiler Desing Lab\Exp7>exp7.exe
123 -45 6.78 -0.9

^Z
1 1 1 1
```

Experiment No:08

Experiment Name: Write a lex program to verify a E-mail address. Ex:mmks735.bd@gmail.com, Mmks#protonmail.com

---

Objectives:

1. To understand the structure and format of a valid email address.
2. To use Flex to write a regular expression pattern for detecting valid email addresses.
3. To create a lexical analyzer that identifies and verifies email format correctness.
4. To validate input strings against the defined email pattern.
5. To print whether the given input is a valid or invalid email address.

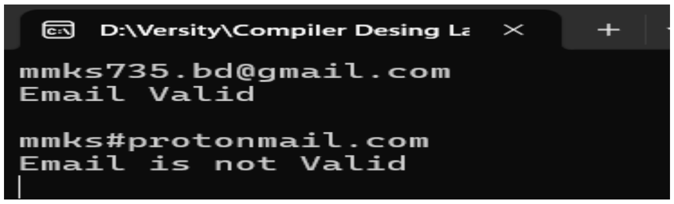
Tools:

• Flex Software
• Windows Terminal (CMD)

Source Code:

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

%%
[a-z0-9._]*[@][a-z.]* {printf("Email Valid\n");}
.* {printf("Email is not Valid");}
%%
int main()
{
yylex();
        return 0;
}
```

Input &Output:



```
mmks735.bd@gmail.com
Email Valid

mmks#protonmail.com
Email is not Valid
```

Experiment No:09

Experiment Name: Write a lex program to make a Calculator App. Ex: 10 + 20 - 5 * 2

Objectives:

1. To learn how to use Lex to recognize and tokenize arithmetic operators and operands.
2. To write patterns in Lex for identifying numbers (integers and floating-point) and mathematical operators (+, -, *, /).
3. To develop a lexical analyzer that can parse arithmetic expressions from input.
4. To implement basic calculation logic by integrating Lex with C code or external functions.
5. To understand how to handle input processing and display the calculated result.
6. To practice building a simple interactive calculator using lexical analysis techniques.
7. To enhance skills in combining pattern matching with computation in Lex.

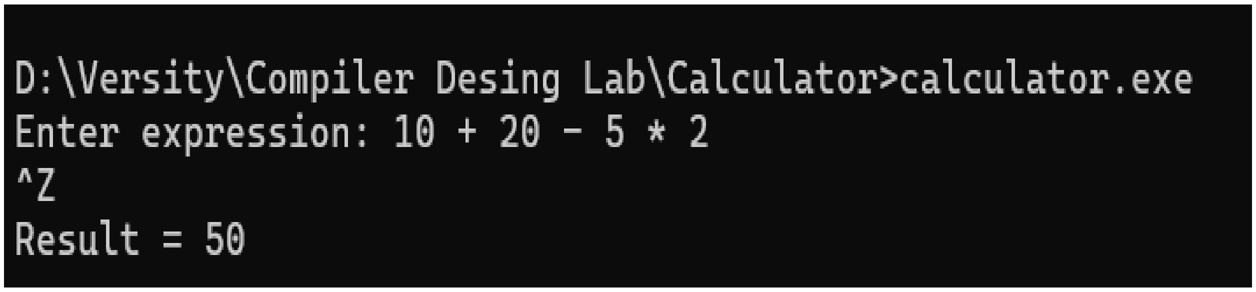
Tools:

• Flex Software
• Windows Terminal (CMD)

Source Code:

```
%option noyywrap
%{
#include <stdio.h>
#include <stdlib.h>

int result = 0;
int last_op = '+';

void calculate(int num) {
    switch(last_op) {
        case '+': result += num; break;
        case '-': result -= num; break;
        case '*': result *= num; break;
        case '/':
            if(num == 0) {
                printf("Error: Division by zero\n");
                exit(1);
            }
            result /= num;
            break;
    }
}
%}

%%
[0-9]+ {
    int num = atoi(yytext);
    calculate(num);
}
[+\-*/] {
    last_op = yytext[0];
}
[ \t\n]+  ;  // ignore whitespace
. {
    printf("Invalid character: %s\n", yytext);
    exit(1);
}
<<EOF>> {
    printf("Result = %d\n", result);
    return 0;
}
%%

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

Input &Output:



```
D:\Versity\Compiler Desing Lab\Calculator>calculator.exe
Enter expression: 10 + 20 - 5 * 2
^Z
Result = 50
```