LEX Program Implementation
Compiler Design – Lab 4

Name: Ojas Patil
Reg No: 21BAI1106

AIM

Write a LEX program to–
a. Count the number of vowels and consonants in a given string.
b. Count the number of keywords, identifiers and digits in the given string/sample C program.
c. Whether given string is a valid VIT mail ID.

Part A – Count the number of Vowels and Consonants in a given String.

Algorithm

1. Initialize counters: Set the counters vowel_count and consonant_count to zero.
2. Scan Input String: Read the input string from the user.
3. Iterate Through Characters: For each character in the input string, check if it is a vowel or a consonant. Increment the respective counter accordingly.
4. Display Results: Output the total count of vowels and consonants.
5. Program returns.

Explaination

This algorithm takes an input string, iterates through each character, and checks whether it is a vowel or a consonant. It counts the occurrences of each and then outputs the total counts of vowels and consonants. This is achieved by utilizing regular expressions in the lexer code to recognize vowels and consonants, updating the counters accordingly, and then displaying the results to the user.

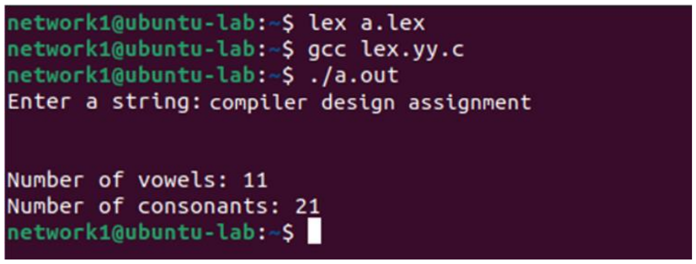
Source Code

```
%{
#include <stdio.h>
int vowel_count = 0;
int consonant_count = 0;
%}
%%
[aeiouAEIOU] { vowel_count++; }
[a-zA-Z] { consonant_count++; }
2 | Page

. ;
%%
int main() {
    char input[100];
    printf("Enter a string: ");
    fgets(input, sizeof(input), stdin);
    yy_scan_string(input);
    yylex();
    printf("\nNumber of vowels: %d\n", vowel_count);
    printf("Number of consonants: %d\n", consonant_count);
    return 0;
}
int yywrap(){return(1);}

// Maam I don't have linux on my system, therefore I have used my roommates system
```

Output Screenshot



```
network1@ubuntu-lab:~$ lex a.lex
network1@ubuntu-lab:~$ gcc lex.yy.c
network1@ubuntu-lab:~$ ./a.out
Enter a string: compiler design assignment


Number of vowels: 11
Number of consonants: 21
network1@ubuntu-lab:~$
```

Part B - Count the number of keywords, identifiers and digits in the given string/sample C program.

Algorithm

1. Initialize Counters: Initialize counters keyword_count, identifier_count, and digit_count to zero.
2. Scan Input: Prompt the user to enter a string or a sample C program.
3. Token Recognition: Parse the input string, recognizing tokens based on predefined patterns.
4. If a token matches any C keyword, increment keyword_count.
5. If a token matches the pattern for an identifier, increment identifier_count.
6. If a token consists only of digits, increment digit_count.
7. Display Results: Output the total counts of keywords, identifiers, and digits.
8. Return: End the program.

Explanation

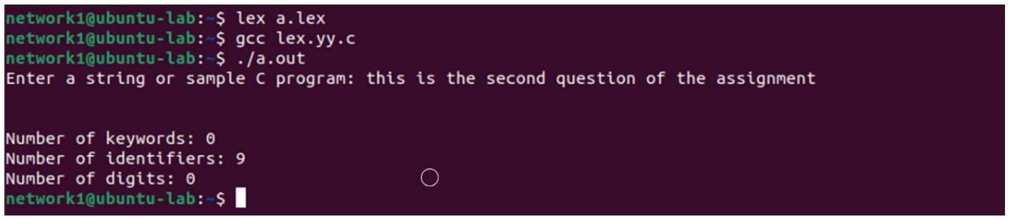
This algorithm takes an input string or a sample C program, tokenizes it, and then counts the occurrences of three types of tokens: keywords, identifiers, and digits. It achieves this by defining patterns for each token type using regular expressions in the lexer code. After parsing the input and updating the counters accordingly, it displays the total counts of keywords, identifiers, and digits to the user. This algorithm enables users to analyze the structure and composition of C programs efficiently.

Source Code

```
%{
#include <stdio.h>
int keyword_count = 0;
int identifier_count = 0;
int digit_count = 0;
%}
%%
"auto"|"break"|"case"|"char"|"const"|"continue"|"default"|"do"|"double
 "|"else"|"enum"|"extern"|"float"|"for"|"goto"|"if"|"int"|"long"|"register
 "|"return"|"short"|"signed"|"sizeof"|"static"|"struct"|"switch"|"typedef"|
 "union"|"unsigned"|"void"|"volatile"|"while"    { keyword_count++; }
[a-zA-Z_][a-zA-Z0-9_]*     { identifier_count++; }
[0-9]+                      { digit_count++; }
.                           ;
```

```
%%
int main() {
    char input[100];
    printf("Enter a string or sample C program: ");
    fgets(input, sizeof(input), stdin);
    yy_scan_string(input);
    yylex();
    printf("\nNumber of keywords: %d\n", keyword_count);
    printf("Number of identifiers: %d\n", identifier_count);
    printf("Number of digits: %d\n", digit_count);
    return 0;
}
int yywrap(){return(1);}
// Maam I don’t have linux on my system, therefore I have used my roommates
system
```

<u>Output Screenshot</u>



```
network1@ubuntu-lab:~$ lex a.lex
network1@ubuntu-lab:~$ gcc lex.yy.c
network1@ubuntu-lab:~$ ./a.out
Enter a string or sample C program: this is the second question of the assignment


Number of keywords: 0
Number of identifiers: 9
Number of digits: 0
network1@ubuntu-lab:~$
```

Part C - Whether given string is a valid VIT mail ID.

<u>Algorithm</u>

1. Tokenize Input: Take input string to tokenize.
2. Pattern Matching: Match the input against the pattern for a valid VIT mail ID.
3. If the input matches the pattern, output "Valid VIT mail id" and terminate.
4. If not, output "Invalid VIT mail id" and terminate.
5. Return: End the program.

<u>Explanation</u>

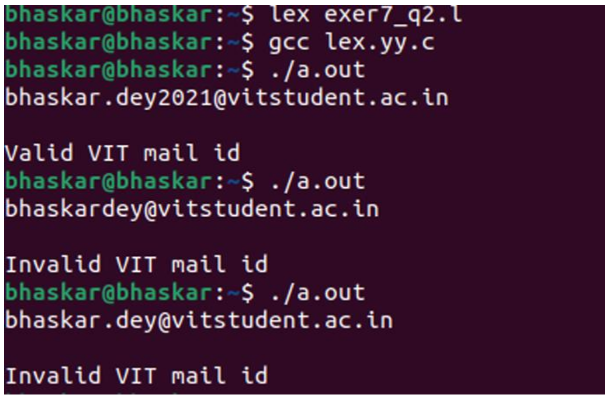
This algorithm verifies whether the input string represents a valid VIT mail ID. It employs a pattern recognition approach, using regular expressions to define the structure of a valid VIT mail ID. If the input matches this pattern, it indicates a valid VIT mail ID and terminates. Otherwise, it signifies an invalid VIT mail ID and terminates. This algorithm provides a simple and efficient method to validate VIT mail IDs.

Source Code

```
%{
%}
%%
[a-z]+[.][a-z]+[2][0][2][1][@][v][i][t][s][t][u][d][e][n][t][.][a][c][.][i][n] {
printf("\nValid VIT mail id\n");
return 0;
}
. {printf("\nInvalid VIT mail id\n");return 0;}
%%
int main(){
yylex();
return 0;
}
int yywrap(){}

// Maam I don’t have linux on my system, therefore I have used my roommates
system
```

Output Screenshot



```
bhaskar@bhaskar:~$ lex exer7_q2.l
bhaskar@bhaskar:~$ gcc lex.yy.c
bhaskar@bhaskar:~$ ./a.out
bhaskar.dey2021@vitstudent.ac.in

Valid VIT mail id
bhaskar@bhaskar:~$ ./a.out
bhaskardey@vitstudent.ac.in

Invalid VIT mail id
bhaskar@bhaskar:~$ ./a.out
bhaskar.dey@vitstudent.ac.in

Invalid VIT mail id
```

Result

Thus, we have created LEX programs for counting vowels and consonants, counting keywords, identifiers etc. and checked whether the entered ID is a valid VIT mail ID. All these programs have been successfully compiled and executed.