NOTRE DAME UNIVERSITY

BANGLADESH

CSE-3104 LAB Report-01

Submitted by:

Name: Istiak Alam

ID: 0692230005101005

Batch: CSE-20

Date: 08-09-24

Course Title: Compiler Design Lab

Course Code: CSE-3104

Lab Task Topic: Compiler Problem Solving using Lex

Submitted To:

Khorshed Alam
Lecturer, NDUB

<u>Problem No. 1:</u> Write a lex program that accept the verbs [am, is, are, was, were]

<u>Solution:</u>

Code : –

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

%%
[/t]+
is |
am |
are |
was |
were        {printf(“ %s : is a verb”, yytext);}
[a-zA-Z]+   {printf(“%s : is not a verb”, yytext);}
%%

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

Code Processing –

Step 1 : Open terminal in Linux and create a lex file named **verb.l**

```
> touch verb.l
```

Step 2 : After Writing the code save it and type command in terminal :

```
> lex verb.l
```

Step 3 : It will create a lex.yy.c file. Then next type command in terminal :

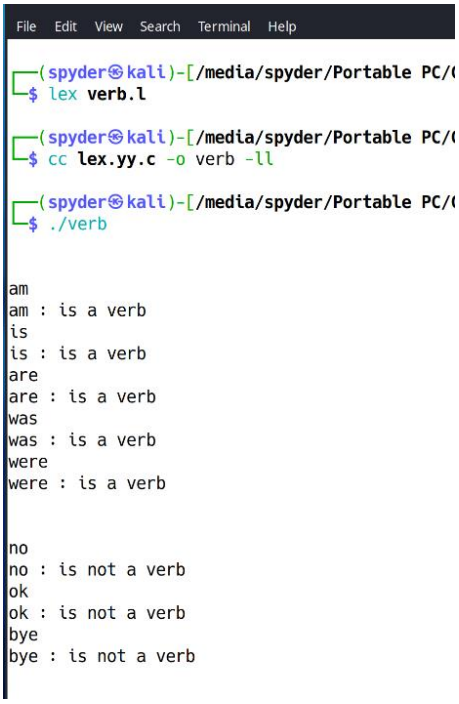
```
> cc lex.yy.c -o verb -ll
```

Step 4 : Then it will create an executable file named **verb.exe**.

Step 5 : Next type command in terminal and run the exe file :

```
> ./verb
```

Output & Explanation:



This program is divided into 3 parts.

- Header section,
- Rules section,
- Main function.

1. The %{ %} block is used to include C/C++ code that should be copied directly into the generated C file.
2. Here, #include <stdio.h> is included, which allows the program to use standard I/O functions such as printf.
3. This %% is a delimiter that separates the definitions section from the rules section. In this case, there's no specific code or definitions provided before the rules section, so it directly starts with the rules.
4. In the rules section, this section contains patterns (regular expressions) and the corresponding actions that should be executed when a pattern is matched.
5. The next lines is | am | are | was | were are patterns that match the specific verbs you are interested in. If any of these words are encountered, the program executes the action specified: {printf(" %s : is a verb", yytext);}.
6. yytext is a built-in variable in Lex that contains the matched text.
7. The printf function prints the matched verb followed by the message is a verb.
8. The second %% marks the end of the rules section and the beginning of user code, which is typically the main function.
9. Inside main(), the yylex() function is called. yylex() is automatically generated by Lex and is responsible for reading the input and applying the rules you've defined. When a match is found, the corresponding action is executed.

<u>Problem No. 2:</u> Write a lex Program that gives output in Start and Stop word.

<u>Solution:</u>

Code : –

```
%{
#include <stdio.h>
%}
start       printf("Start Command received\n");
stop        printf("Stop Command received\n");
%%
```

Code Processing –

Step 1 : Open terminal in Linux and create a lex file named **start.l**

```
> touch start.l
```

Step 2 : After Writing the code save it and type command in terminal :

```
> lex start.l
```

Step 3 : It will create a lex.yy.c file. Then next type command in terminal :

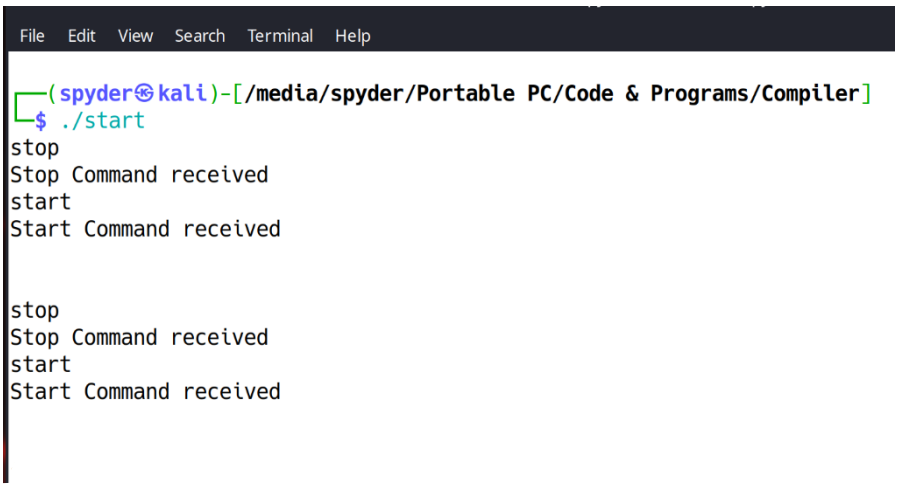
```
> cc lex.yy.c -o start -ll
```

Step 4 : Then it will create an executable file named **start.exe**.

Step 5 : Next type command in terminal and run the exe file :

```
> ./start
```

<u>Output & Explanation:</u>



This Lex program is designed to recognize the words "start" and "stop" in each text and produce corresponding outputs. The program is structured as follows:

- **Header Section**: Includes necessary libraries.
- **Definition Section**: Initially attempted to define macros for start and stop commands.
- **Rules Section**: Should contain patterns to match the words "start" and "stop" and print the appropriate messages.

Explanation :-

1. In the Header Section the %{ %} block is used to include C/C++ code that should be copied directly into the generated C file.
2. Here, #include <stdio.h> is included, which allows the program to use standard I/O functions like printf.
3. In the definition section, two macros or symbolic names are defined: start and stop.
4. The macro start is defined to execute the command printf("Start Command received"); whenever the word "start" is encountered in the input.
5. Similarly, the macro stop is defined to execute the command printf("Stop Command received"); whenever the word "stop" is encountered in the input.
6. The %% marks the beginning of the rules section. Typically, this is where you would write the patterns and corresponding actions.
7. This section should include patterns for matching the start and stop commands and associate them with the defined actions.
8. To make the provided code functional, you need to move the start and stop macros into the rules section and associate them with patterns
9. Now, when the program is run, it will scan the input text. If it encounters the word "start", it will print "Start Command received". If it encounters the word "stop", it will print "Stop Command received".

<u>Problem No. 3:</u> Write a lex Program to count number of characters & number of lines.

<u>Solution:</u>

Code : –

```
%{
#include <stdio.h>
int num_lines=0, num_char=0;
%}

%%
\n          {++num_lines; ++num_char;}
.           {++num_char;}
out         {printf("there are %d lines and %d
characters\n", numb_lines,num_char);}

%%

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

Code Processing –

Step 1 : Open terminal in Linux and create a lex file named **counter.l**

```
> touch counter.l
```

Step 2 : After Writing the code save it and type command in terminal :

```
> lex counter.l
```

Step 3 : It will create a lex.yy.c file. Then next type command in terminal :

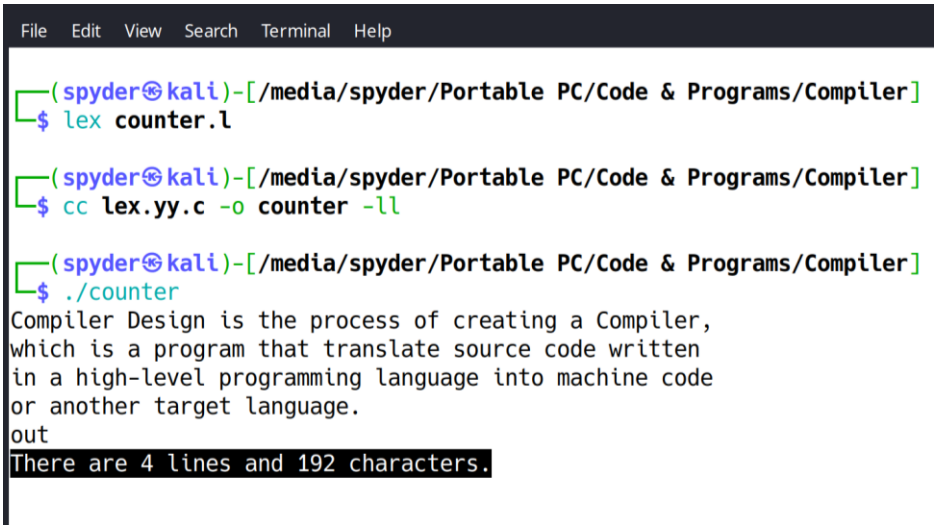
```
> cc lex.yy.c -o counter-ll
```

Step 4 : Then it will create an executable file named **counter.exe**.

Step 5 : Next type command in terminal and run the exe file :

```
> ./counter
```

Output & Explanation:



Explanation :-

1. The %{ %} block is used to include C/C++ code that should be copied directly into the generated C file.
2. Two global variables are defined: num_lines and num_char.
3. The %% marks the beginning of the rules section where patterns and their corresponding actions are defined.
4. This patten **\n** matches a newline character in the input.
5. ++num_lines; increments the num_lines counter by 1, counting the new line.
6. ++num_char; increments the num_char counter by 1, counting the newline character as part of the total characters.
7. This pattern '**.**' matches any single character in the input except a newline.
8. The action {++num_char;} increments the num_char counter by 1, counting each character. This pattern **out** matches the word "out"
9. The action {printf("there are %d lines and %d characters\n", num_lines, num_char);} prints the current count of lines and characters.
10. It outputs the number of lines stored in num_lines and the number of characters stored in num_char when the word "out" is encountered.
11. The second %% marks the end of the rules section and the beginning of the user-defined code section, which typically contains the main function. Inside the main() function, the yylex() function is called. yylex() is automatically generated by Lex and is responsible for scanning the input, applying the defined patterns, and executing the corresponding actions.