

Experiment No. 2

Aim: To implement: 1) Identify a given no is even or odd

2) Identify vowels in a string

3) Identify relational operators

4) Copy content of one file to another

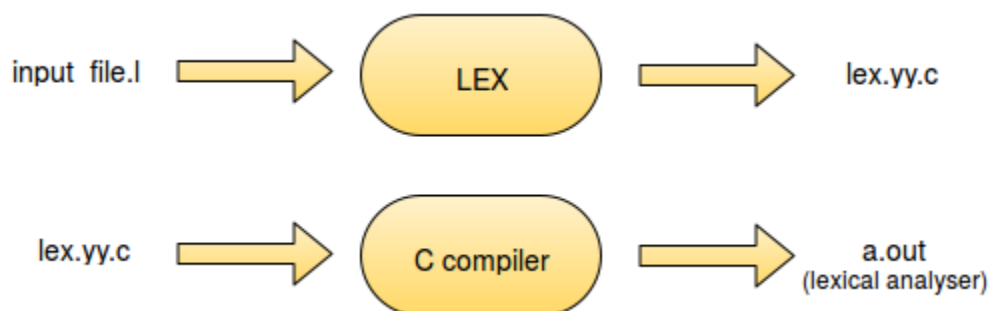
Using Lexical Analyzer tool: LEX

Requirements: Lex tool and C.

Theory:

Introduction

LEX is a tool used to generate a lexical analyzer. This document is a tutorial for the use of LEX for **ExpL Compiler** development. Technically, LEX translates a set of regular expression specifications (given as input in input_file.l) into a C implementation of a corresponding finite state machine (lex.yy.c). This C program, when compiled, yields an executable lexical analyzer.



The source ExpL program is fed as the input to the lexical analyzer which produces a sequence of tokens as output. (Tokens are explained below). Conceptually, a lexical analyzer scans a given source ExpL program and produces an output of tokens.

Each token is specified by a token name. The token name is an abstract symbol representing the kind of lexical unit, e.g., a particular keyword, or a sequence of input characters denoting an identifier. The token names are the input symbols that the parser processes. For instance integer, boolean, begin, end, if, while etc. are tokens in ExpL.

```
“integer”      {return ID_TYPE_INTEGER;}
```

This example demonstrates the specification of a **rule** in LEX. The rule in this example specifies that the lexical analyzer must return the token named ID_TYPE_INTEGER when the pattern “integer” is found in the input file. A rule in a LEX program comprises of a 'pattern' part (specified by a regular expression) and a corresponding (semantic) 'action' part (a sequence of C statements). In the above example, “integer” is the pattern and {return ID_TYPE_INTEGER;} is the corresponding action. The statements in the action part will be executed when the pattern is detected in the input.

Lex was developed by Mike Lesk and Eric Schmidt at Bell labs.

The structure of LEX programs

A LEX program consists of three sections: **Declarations, Rules and Auxiliary functions**

DECLARATIONS

```
%%
```

RULES

```
%%
```

AUXILIARY FUNCTIONS

1 Declarations

The declarations section consists of two parts, **auxiliary declarations** and **regular definitions**.

The auxiliary declarations are copied as such by LEX to the output *lex.yy.c* file. This C code consists of instructions to the C compiler and are not processed by the LEX tool. The auxiliary declarations (which are optional) are written in C language and are enclosed within '%{ ' and '%} '. It is generally used to declare functions, include header files, or define global variables and constants.

LEX allows the use of short-hands and extensions to regular expressions for the regular definitions. A regular definition in LEX is of the form: **D R** where D is the symbol representing the regular expression R.

2 Rules

Rules in a LEX program consists of two parts:

1. The pattern to be matched
2. The corresponding action to be executed

3 Auxiliary functions

LEX generates C code for the rules specified in the Rules section and places this code into a single function called *yylex()*. (To be discussed in detail later). In addition to this LEX generated code, the programmer may wish to add his own code to the *lex.yy.c* file. The auxiliary functions section allows the programmer to achieve this.

The auxiliary declarations and auxiliary functions are copied as such to the *lex.yy.c* file

Once the code is written, *lex.yy.c* maybe generated using the command `lex "filename.l"` and compiled as `gcc lex.yy.c`

1) Even Odd

Code:

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

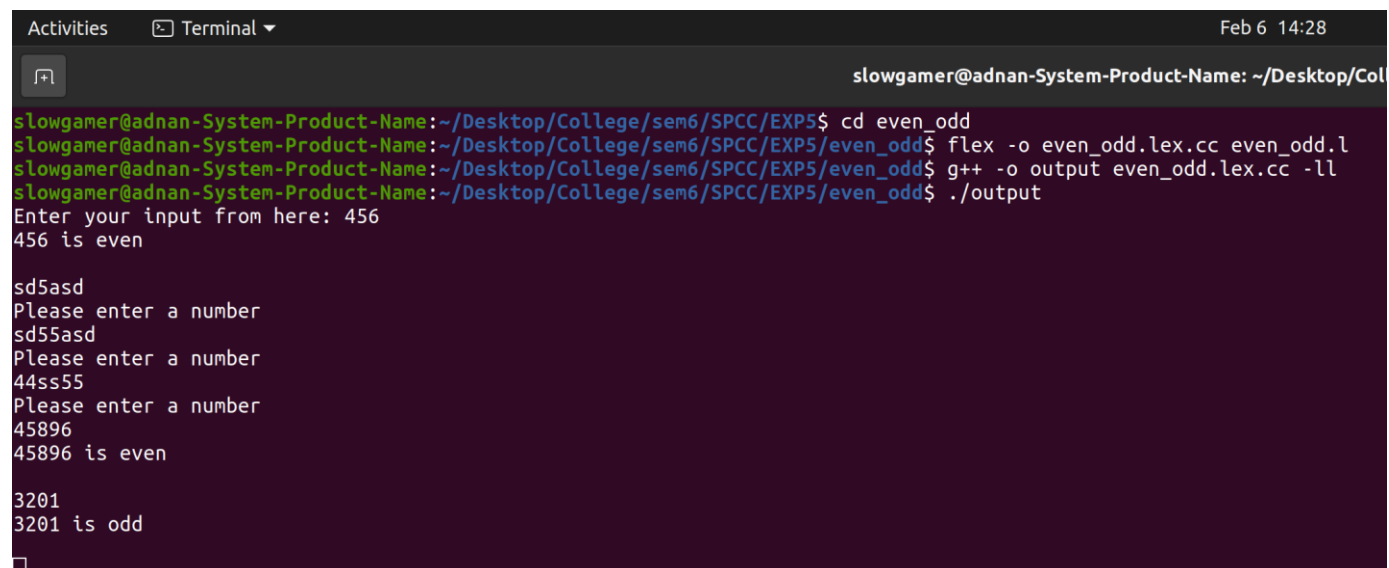
^[0-9]+$ {
    if(atoi(yytext)%2) printf("%s is odd\n",yytext);
    else printf("%s is even\n",yytext);
}
[a-zA-Z0-9_]+ {printf("Please enter a number");}

%%

int yywrap(){
    return 0;
}

int main(){
    printf("Enter your input from here: ");
    yylex();
}
```

Output:



```
slowgamer@adnan-System-Product-Name: ~/Desktop/Col
slowgamer@adnan-System-Product-Name:~/Desktop/College/sem6/SPCC/EXP5$ cd even_odd
slowgamer@adnan-System-Product-Name:~/Desktop/College/sem6/SPCC/EXP5/even_odd$ flex -o even_odd.lex.cc even_odd.l
slowgamer@adnan-System-Product-Name:~/Desktop/College/sem6/SPCC/EXP5/even_odd$ g++ -o output even_odd.lex.cc -ll
slowgamer@adnan-System-Product-Name:~/Desktop/College/sem6/SPCC/EXP5/even_odd$ ./output
Enter your input from here: 456
456 is even

sd5asd
Please enter a number
sd55asd
Please enter a number
44ss55
Please enter a number
45896
45896 is even

3201
3201 is odd
```


3) Relational Operator

Code:

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

%%
- {printf("- operator");}
\ / {printf("/ operator");}
\+ {printf("+ operator");}
\* {printf("* operator");}
\> {printf("> operator");}
\< {printf("< operator");}
\>= {printf(">= operator");}
\<= {printf("<= operator");}
= {printf("=" operator");}
== {printf("== operator");}
[0-9]+ {printf("Number");}
[A-Za-z][A-Za-z0-9_]* {printf("Identifier");}
.* {printf("Cannot be identified");}

%%
int main()
{
    printf("Enter operator or operand: ");
    yylex();
    return 0;
}
```

Output:

```
C:\WINDOWS\system32\cmd.exe - relational

C:\Users\adnan\OneDrive\Desktop\College\sem6\SPCC\EXP2\relational>relational
Enter operator or operand: asdas56
Identifier
546
Number
+
+ operator
\
Cannot be identified
/
/ operator
>=
>= operator
<
< operator
<=
<= operator
,
Cannot be identified
<
< operator
```

4) Moving Content of a file

Code:

```
% {
int nlines,nwords,nchars;
% }

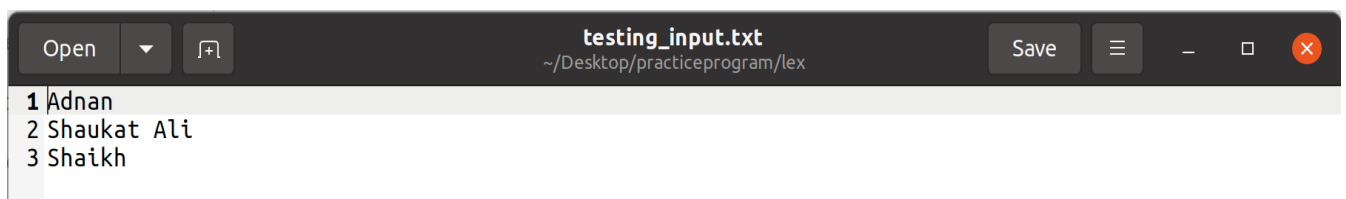
%%
\n {
    nchars++;nlines++; fprintf(yyout,"%s",yytext);
}

[^\n\t]+ {nwords++; nchars=nchars+yyleng; fprintf(yyout,"%s",yytext);}
. {nchars++; fprintf(yyout,"%s",yytext);}
%%
int yywrap(void)
{
    return 1;
}
int main(int argc, char*argv[])
{
    yyin = fopen(argv[1],"r");
    yyout = fopen(argv[2],"w");
    yylex();
    fclose(yyin);
    fclose(yyout);
    printf("Lines = %d\nChars=%d\nWords=%d",nlines,nchars,nwords);

    return 0;
}
```

Output:

Input.txt



The screenshot shows a text editor window titled "testing_input.txt" with the path "~/Desktop/practiceprogram/lex". The window contains the following text:

```
1 Adnan
2 Shaukat Ali
3 Shaikh
```

Executing code

```

slowgamer@adnan-System-Product-Name: ~/Desktop/practiceprogram/lex
slowgamer@adnan-System-Product-Name:~$ cd Desktop/practiceprogram/lex
slowgamer@adnan-System-Product-Name:~/Desktop/practiceprogram/lex$ flex -o testing_complie.lex.cc testing.l
slowgamer@adnan-System-Product-Name:~/Desktop/practiceprogram/lex$ g++ -o testing_output testing_complie.lex.cc
g++: error: testing_complie.lex.cc: No such file or directory
g++: fatal error: no input files
compilation terminated.
slowgamer@adnan-System-Product-Name:~/Desktop/practiceprogram/lex$ ls
testing_complie.lex.cc  testing_input.txt  testing.l
slowgamer@adnan-System-Product-Name:~/Desktop/practiceprogram/lex$ g++ -o testing_output testing_complie.lex.cc
slowgamer@adnan-System-Product-Name:~/Desktop/practiceprogram/lex$ ./testing_output
^C
slowgamer@adnan-System-Product-Name:~/Desktop/practiceprogram/lex$ ./testing_output testing_input.txt testing_out
put.txt
Lines = 2
Chars=26
Words=4slowgamer@adnan-System-Product-Name:~/Desktop/practiceprogram/lex$ ^C
slowgamer@adnan-System-Product-Name:~/Desktop/practiceprogram/lex$

```

Output.txt

```

testing_output.txt
~/Desktop/practiceprogram/lex
1 Adnan
2 Shaukat Ali
3 Shaikh

```

Conclusion: We have successfully implemented program to: Identify a given no is even or odd, Identify vowels in a string, Identify relational operators and Copy content of one file to another in LEX.