**Name: Amarsingh Kashyap**

**Roll No: 101**

**Practical No. 1**

**Theory**

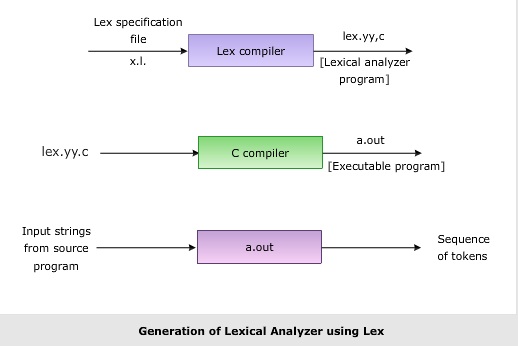
**LEX:**

Lex is a program generator designed for lexical processing of character input streams. It accepts a high level, problem-oriented specification for character string matching, and produces a program in a general purpose language which recognizes regular expressions. The regular expressions are specified by the user in the source specifications given to Lex. The Lex written code recognizes these expressions in an input stream and partitions the input stream into strings matching the expressions. At the boundaries between strings program sections provided by the user are executed. The Lex source file associates the regular expressions and the program fragments. As each expression appears in the input to the program written by Lex, the corresponding fragment is executed.

Lex is not a complete language, but rather a generator representing a new language feature which can be added to different programming languages, called ``host languages.'' Just as general purpose languages can produce code to run on different com puter hardware, Lex can write code in different host languages.

Lex turns the user's expressions and actions (called source in this pic) into the host general-purpose language; the generated program is named yylex. The yylex program will recognize expressions in a stream (called input in this pic) and perform the specified actions for each expression as it is detected.

**Diagram of LEX**



**Format for Lex file**

The general format of Lex source is:

{definitions}

%%

{rules}

%%

{user subroutines}

where the definitions and the user subroutines are often omitted. The second %% is optional, but the first is required to mark the beginning of the rules. The absolute minimum Lex program is thus %% (no definitions, no rules) which translates into a program which copies the input to the output unchanged.

**Regular Expression**

A regular expression (or RE) specifies a set of strings that matches it; the functions in this module let you check if a particular string matches a given regular expression (or if a given regular expression matches a particular string, which comes down to the same thing).

Regular expressions can be concatenated to form new regular expressions; if A and B are both regular expressions, then AB is also a regular expression. In general, if a string p matches A and another string q matches B, the string pqwill match AB. This holds unless A or B contain low precedence operations; boundary conditions between A and B; or have numbered group references. Thus, complex expressions can easily be constructed from simpler primitive expressions.Regular expressions can contain both special and ordinary characters. Most ordinary characters, like "A", "a", or "0", are the simplest regular expressions; they simply match themselves. You can concatenate ordinary characters, so last matches the string 'last'. (In the rest of this section, we'll write RE's in this special style, usually without quotes, and strings to be matched 'in single quotes'.)

Some characters, like "|" or "(", are special. Special characters either stand for classes of ordinary characters or affect how the regular expressions around them are interpreted.

**Lex Library Routines**

Lex library routines are those functions which have a detailed knowledge of the lex functionalities and which can be called to implement various tasks in a lex program.

The following table gives a list of some of the lex routines.

|  |  |
| --- | --- |
| Lex Routine | Description |
| Main() | Invokes the lexical analyzer by calling the yylex subroutine. |
| yywrap() | Returns the value 1 when the end of input occurs. |
| yymore() | Appends the next matched string to the current value of the yytext array rather than replacing the contents of the yytext array. |
| yyless(int n) | Retains n initial characters in the yytext array and returns the remaining characters to the input stream. |
| yyreject | Allows the lexical analyzer to match multiple rules for the same input string. (The yyreject subroutine is called when the special action REJECT is used.) |
| yylex() | The default main() contains the call of yylex() |

**Answer the Questions:**

1. Use of yywrap

Ans: Returns the value 1 when the end of input occurs.

1. Use of yylex function

Ans: The default main() contains the call of yylex()

1. What does lex.yy.c. do ?

Ans: This is a command used to compile a LEX file.

**Practical No. 1 (A)**

**Aim :**

**Use the above code (S1) and perform the additional tasks: If a keyword is found append AAA to the identified keyword. For identifier append III. Also add 2 to digit and display the answer.**

**Program:**

**%{**

**#include<stdio.h>**

**#include<string.h>**

**#include<stdlib.h>**

**char str[100];**

**%}**

**integer [0-9]**

**letter [a-zA-Z]**

**identifier [{letter}|"\_"][{letter}|{integer}|"\_"]\***

**operator "+"|"-"|"\*"|"/"|"%"|"++"|"--"|"=="|"!="|">"|"<"|"<="|">="|"&&"|"||"|"!"|"="|"+"|"-="|"\*="|"%="|"/="**

**keyword "do"|"if"|"else"|"while"|"for"|"int"|"float"|"char"|"double"|"void"|"return"|"case"|"break"|"continue"|"switch"|"printf"**

**string \".\*\"**

**%%**

**{integer} {printf("[int: %s-> %d]",yytext, atoi(yytext)+2);}**

**{keyword} { strcpy(str, yytext);**

**strcat(str,"AAA");**

**printf("[kw: %s]", str);**

**}**

**{operator} {printf("[op: %s]", yytext);}**

**[A-Za-z\_][A-Za-z0-9\_]\* { strcpy(str, yytext);**

**strcat(str, "III");**

**printf("[id: %s]", str);**

**}**

**{string} {printf("[str: %s]", yytext);}**

**%%**

**int yywrap(void)**

**{**

**}**

**int main()**

**{**

**yylex();**

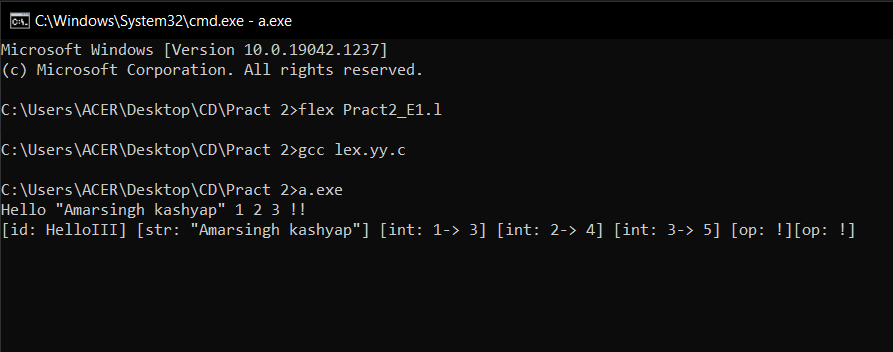
**return 0;**

**}**

**Input:**

**Hello "Amarsingh kashyap" 1 2 3 !!**

**Output:**



**Practical No. 1 (B)**

**Aim:**

**Write a LEX specification to take the contents from a file while adding 3 to number divisible by 7 and adding 4 to number divisible by**

**Program:**

**%{**

**#include<stdio.h>**

**#include<stdlib.h>**

**#include<string.h>**

**int a;**

**%}**

**%%**

**[0-9]+ {a = atoi(yytext);**

**printf("%d", a);**

**if (a%7 == 0)**

**printf("\t[Divisible by 7]+3 -> %d", (a+3));**

**if (a%2 == 0)**

**printf("\t[Divisible by 2]+2 -> %d", a, (a+2));**

**}**

**%%**

**int yywrap(void) {}**

**int main() {**

**yylex();**

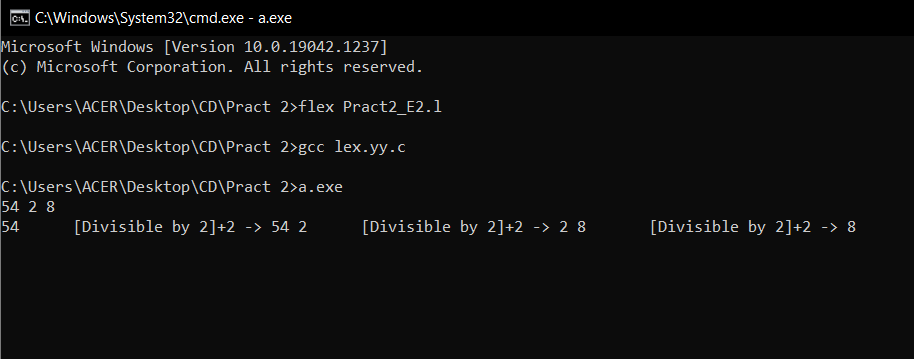
**return 0;**

**}**

**Input:**

**54 2 8**

**Output:**



**Practical No. 1 (C)**

**Aim:**

**Write a lex specification to display the histograms of length of words**

**Program:**

**%{**

**#include<stdio.h>**

**#include<stdlib.h>**

**#include<string.h>**

**int count[50];**

**int l = 0;**

**%}**

**%%**

**[A-Za-z0-9]\* {count[yyleng]+=1;}**

**%%**

**int yywrap(void)**

**{**

**}**

**int main()**

**{**

**yylex();**

**int i = 0;**

**printf("Frequency of word are : ");**

**for( i = 0; i<50; i++)**

**{**

**if(count[i]>0)**

**{**

**printf("\n\t%d\t\t%3d", i, count[i]);**

**}**

**}**

**printf("\n");**

**memset(count, 0, 50);**

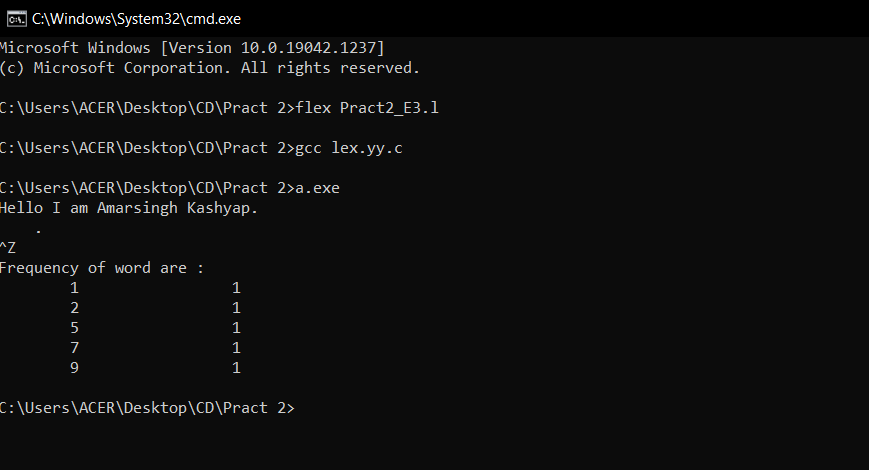
**return 0;**

**}**

**Input:**

**Hello I am Amarsingh Kashyap.**

**Output:**



**Practical No. 1 (D)**

**Aim:**

**Write a LEX specification to search the input file. Let the input file contain some text,**

**comments and digits.**

**(i) Convert text present in file to LOWERCASE.**

**(ii) Report occurrence of comments and special characters.**

**Program:**

**lower [a-z]**

**upper [A-Z]**

**digit [0-9]**

**%{**

**#include<stdio.h>**

**#include<string.h>**

**int comments = 0, spl\_char = 0;**

**%}**

**%%**

**{upper} {**

**printf("%c", (int)yytext[0] + 32);**

**}**

**"//".\*\n {comments++;}**

**"/\*".\*"\*/"\n {comments++;}**

**[^\n {upper}{lower}{digit}] {spl\_char++;}**

**%%**

**main()**

**{**

**yyin = fopen("input4.txt","r");**

**yylex();**

**printf("\nComments : %d\n", comments);**

**printf("\nSpecial Character : %d\n", spl\_char);**

**}**

**int yywrap()**

**{**

**return(1);**

**}**

**Input:**

**Input file Content**

**-------------------------------------------------------------------------**

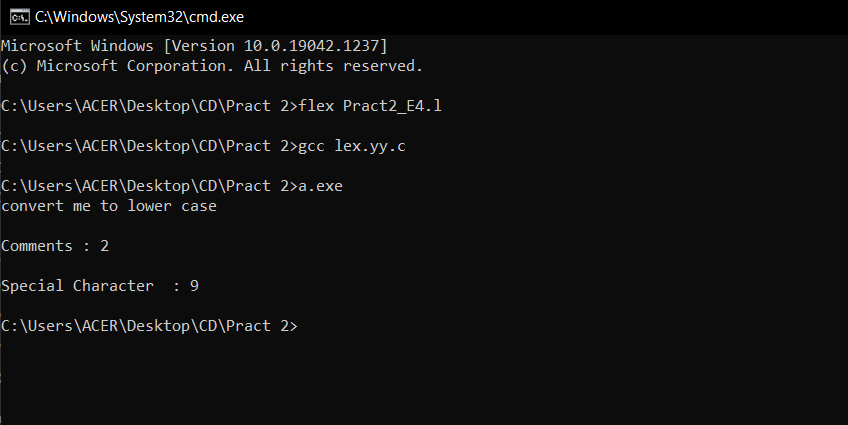
**CONVERT ME TO LOWER CASE**

**/\*Multiline comment\*/**

**//SINGLE LINE comment**

**#@\*^&###**

**Output:**



**Practical No. 1 (E)**

**Aim:**

**Translate an HTML file with some HTML tags to text file using lex. Consider input from**

**stdin. Discard all HTML tags and comments and write the remaining text to stdout. The text output should simulate the HTML characteristics such as list, indent and paragraphs. Font characters such as bold and italics may not be simulated.**

**Program:**

**%{**

**%}**

**%%**

**" "[^>]\*" " {printf("%s\n", yytext); }**

**. ;**

**%%**

**int yywrap(){}**

**int main(int argc, char\*argv[])**

**{**

**// Open tags.txt in read mode**

**yyin = fopen("input5.txt","r");**

**// The function that starts the analysis**

**yylex();**

**return 0;**

**}**

**Input:**

**Input file Content**

**<p> This is a paragraph! </p>**

**</br>**

**<ol> Shopping list**

**<li> Bread </li>**

**<li> Butter </li>**

**<li> Magnets </li>**

**<li> Paint Brush </li>**

**</ol>**

**</br>**

**<ul> TODO list**

**<li> Call Customer Care </li>**

**<li> Collect Parcel </li>**

**<li> Download Virtual Box </li>**

**<li> Drafts Tasks </li>**

**</ul>**

**Output:**

