**Practical No. 1**

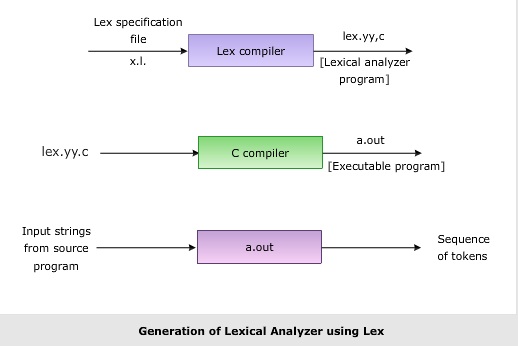
**LEX:**

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 ‘pq’ will 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() |

**Additional TASK:**

**ADD DATA from Lex presentation**

**Also add answers to following Qs:**

1. **Why is –ll option used for running lex.yy.c**

-ll option is used to link the output file to the program file

**2. Use of yywrap**

**Ans.** Function yywrap is called by lex when input is exhausted. Return 1 if you are done or 0 if more processing is required. Every C program requires a main function. In this case we simply call yylex that is the main entry-point for lex. Some implementations of lex include copies of main and yywrap in a library thus eliminating the need to code them explicitly. This is why our first example, the shortest lex program, functioned properly. When the end of the file is reached the return value of yywrap() is checked. If it is non-zero, scanning terminates and if it is 0 scanning continues with next input file.

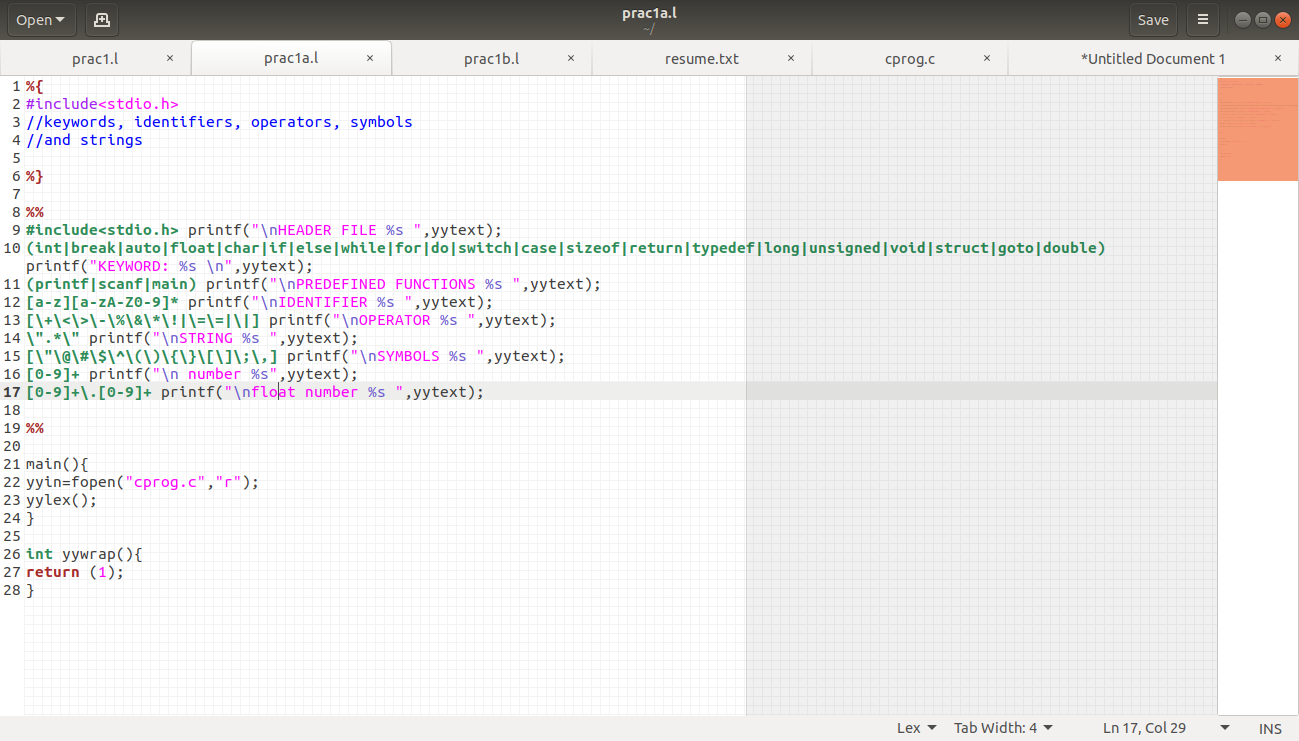
**3. Internal representation of Lex**

Lex internally uses a state machine to parse and identify the tokens from the input file. These programs perform character parsing and tokenizing via the use of a deterministic finite automaton (DFA). A DFA is a theoretical machine accepting regular languages. These machines are a subset of the collection of Turing machines. DFAs are equivalent to read-only right moving Turing machines. The syntax is based on the use of regular expressions.

**Practical 1 -A**

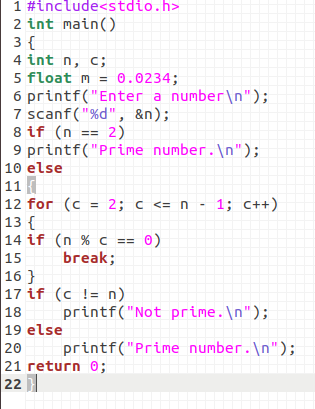
A) Design a lexical analyser to identify the tokens such as keywords, identifiers, operators, symbols and strings for C language using Lex.

*Lex file:*



----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

*C file(Input)*



---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

*Output:*

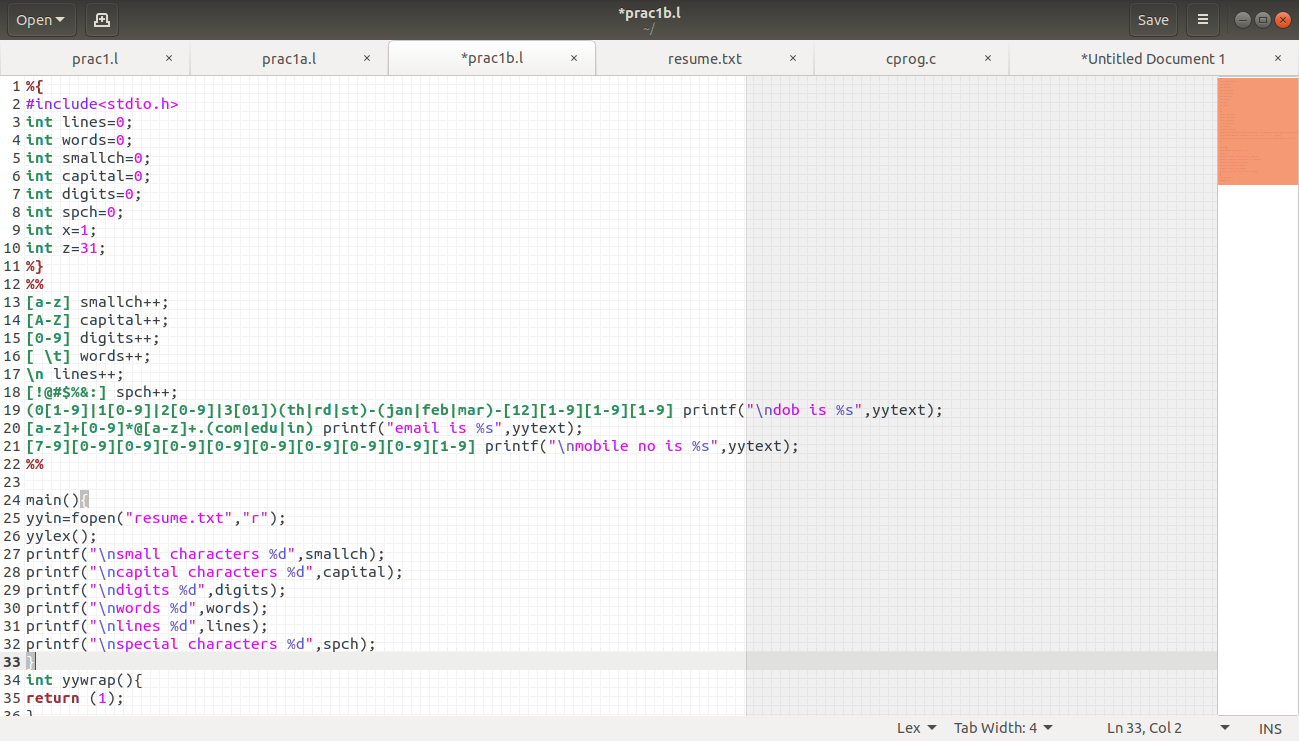
**

**Practical 1 -B**

A)Write a Lex program to find the parameters given below. Consider as input a resume of the student and find:

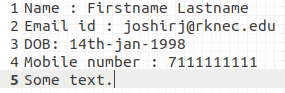
date of birth, mail ID, mobile number, number of words, lines, small letters, capital letters, digits(integer & floating point) and special characters etc.

*Lex file:*



----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

*file(Input)*

-

--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

*Output:*

