**Name: Tina Borundia**

**Roll No.C1-16**

**Practical No. 1**

**Theory**

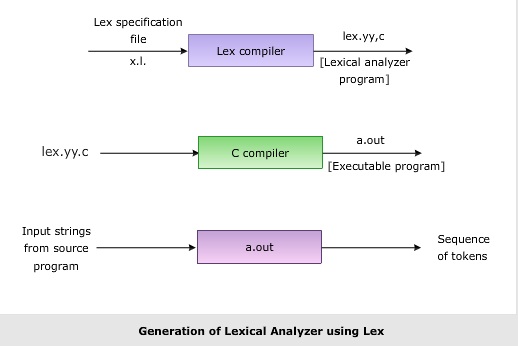
**LEX:**

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:

yywrap is a function in **Lex** (or Flex) that is called by the yylex function when the end of the input file is reached. It determines whether Lex should continue scanning another input file or terminate processing. By default, yywrap is defined to always return 1, indicating that scanning should stop when the end of the input is reached. You can override yywrap to return 0 if you want to switch to a new input file and continue scanning.

1. Use of yylex function

yylex is the main lexical analyzer function automatically generated by Lex (or Flex) based on the patterns and actions defined in the .l file. It reads the input, matches it against the patterns defined in the Lex file, and executes the corresponding actions. It

scans input data character by character. Identifies tokens by matching patterns in the input to the rules in the Lex file. Returns the identified tokens to the parser or main program for further processing.

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

lex.yy.c is the C source code file generated by the Lex (or Flex) tool after processing a .l (Lex specification) file. It contains the code for the lexical analyzer, including:

* The yylex function implementation.
* Tables and logic for matching patterns defined in the Lex file.
* Actions associated with each matched pattern.

The generated lex.yy.c file is compiled into an object file and linked with other parts of the program (e.g., a parser) to build the complete executable.

**Practical No. E1**

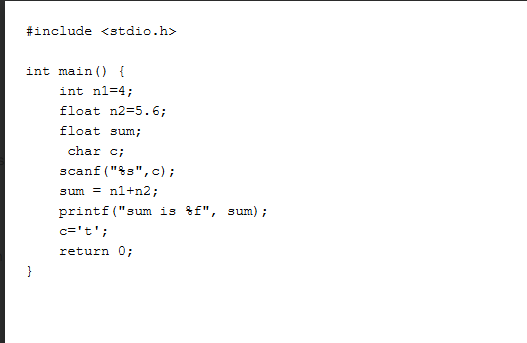
**Aim :Design a lexical analyzer**

Write a lex code to identify the tokens such as keywords, identifiers, operators, constants (Int, float & character), special symbols and strings for C language using LEX. Use File for the input

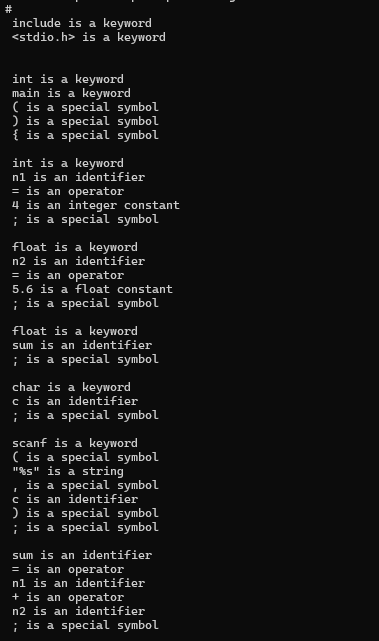
**Program:**

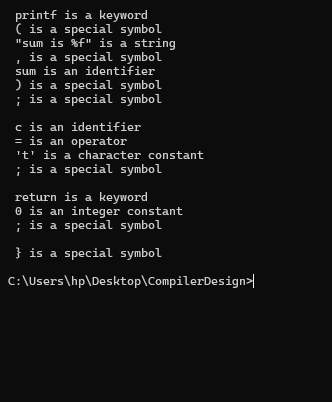
****

**Input:**

****

**Output:**

****

****

**Practical No. E2**

**Aim: Question Paper Analyzer**

Write a Lex program to find the parameters given below. Consider as input a question paper of an examination.

1. Count the number of questions.
2. Number of questions that have sub-part and how many donot.
3. Count the total marks.
4. Date of examination
5. Semester
6. Count different types of questions- Eg: What, Discuss, etc.
7. Numbers of words, lines, small letters, capital letters, digits, and special characters.

**Sample Question paper:**

Shri Ramdeobaba College of Engineering and Management

Sem: VI

12/12/2024

Solve as per questions.

Q1(a): What is a compiler? [5Marks]

Q1(b): What is a software? [5Marks]

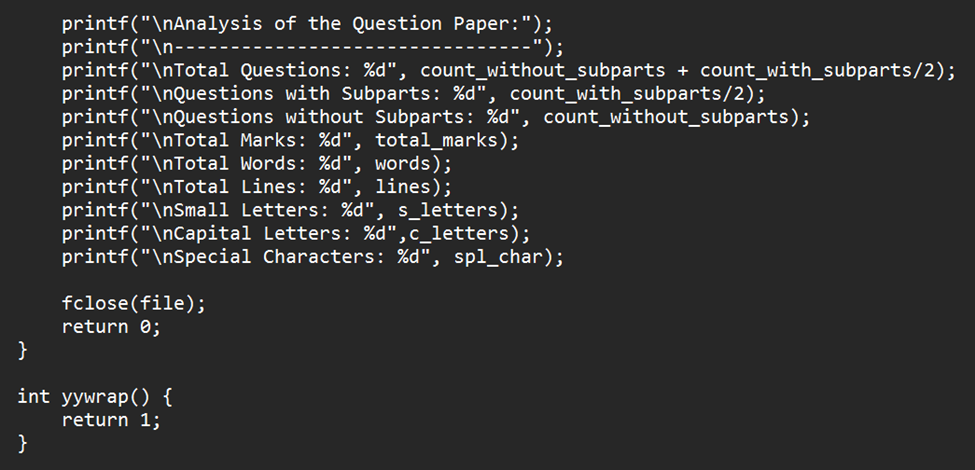
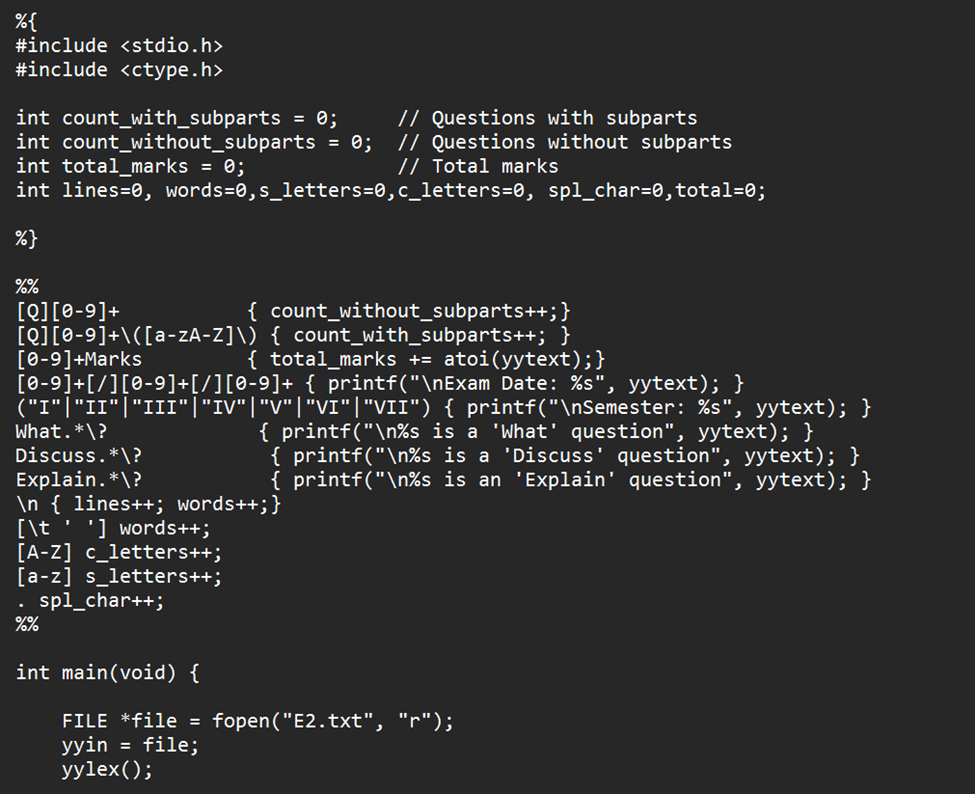
Q2(a): Explain the need for an assembler. [3Marks]

Q2(b): Discuss phases of compiler. [2Marks]

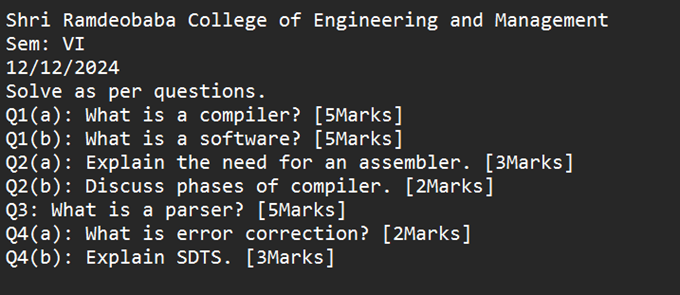
Q3: What is a parser? [5Marks]

Q4(a): What is error correction? [2Marks]

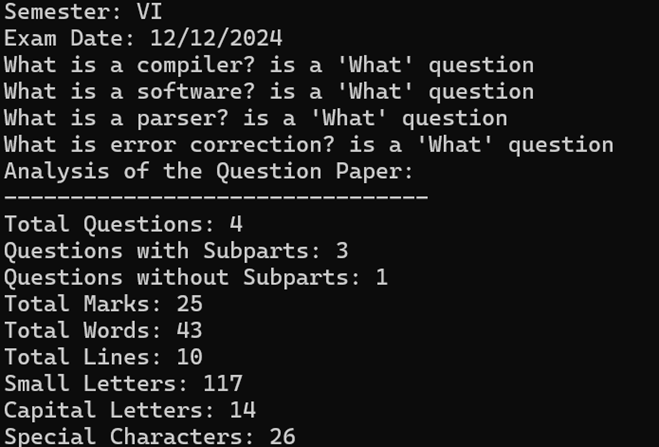
Q4(b): Explain SDTS. [3Marks]

**Program:** 

**Input:**



**Output:**



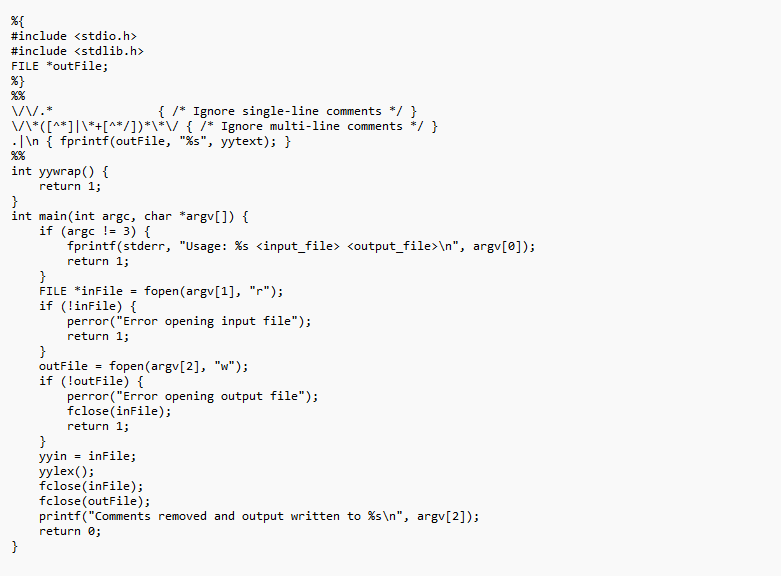
**Practical No. E3**

**Aim: Program Cleaner**

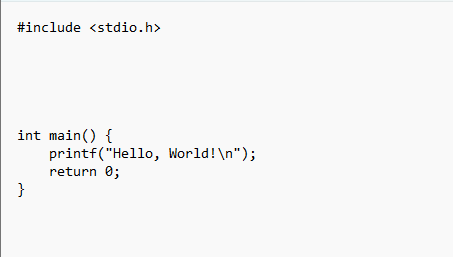
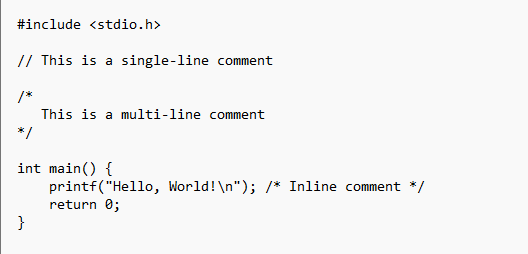
Write a Lex Program which takes C program from file and write the same C program in

another file after removing the comments.

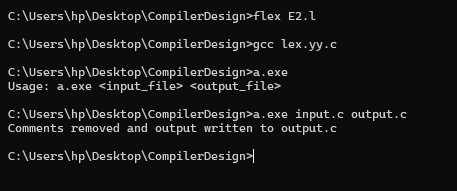
**Program:**



**Input:**



**Output:**



**Practical No. E4**

**Aim: Do as directed**

Write a LEX specification to take the contents from a file

1. Add 3 to number divisible by 7
2. Add 4 to number divisible by 2
3. Convert the alphabetical list to numbered list

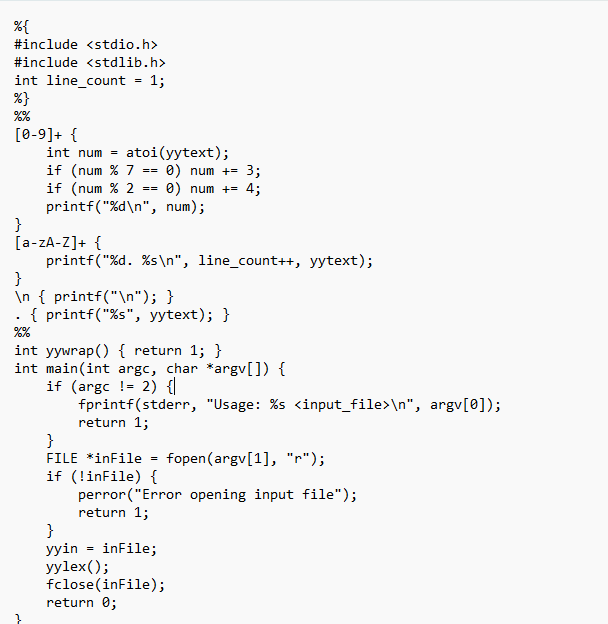
**Sample Input:**

14

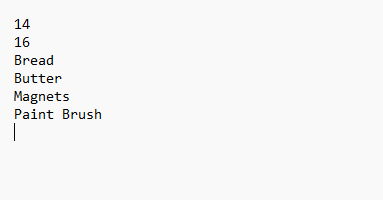
16

1. Bread
2. Butter
3. Magnets
4. Paint Brush

**Program:**



**Input:**



**Output:**

