Name: Prathamesh Prabhakar Thakare
Class: TY CS D
Batch: 2
Roll number: 58
PRN: 12220016

# Assignment number 3: Write a Lex Program to Convert Cases Of Given Data.

**Theory:**

**Lexical Analysis and Lex:**
Lex is a powerful tool used to create lexical analyzers or scanners, which are essential for processing and tokenizing text or source code.

**Structure of Lex Code**
Lex code is typically organized into three main sections:

Definitions Section (%{ ... %}):

In this section, you can include C code that will be directly incorporated into the generated lexer code. For example, you can define variables and functions here that you want to use throughout your lexer.

Rules Section (%% ... %%):

This section is where you define patterns and their corresponding actions.
These patterns are used to identify and process tokens from the input text.
Actions specify what should be done when a particular pattern is matched.
These patterns and actions together constitute the core logic of the lexer.

User Code Section:

Typically, the int main() function serves as the entry point for the generated lexer code.
In this section, you invoke yylex() to initiate the scanning process on the input text.
The user code section can include any additional code or functions required to interact with the lexer or handle the results.

**Definitions Section Example:**
- Within the definitions section, you can define variables and functions that assist in the lexer's operation.
- For instance, you might declare variables to keep track of counts or other state information.

**Rules Section Example:**
- In the rules section, you define regular expressions or patterns that specify the tokens you want to identify.
- For each pattern, you provide an associated action or code snippet that is executed when the pattern is matched.
- Patterns and actions collectively define how the lexer tokenizes and processes the input.

**User Code Section Example:**
- The main() function in the user code section is where the lexer's execution begins.
- It often involves setting up the input source, invoking yylex() to start scanning, and possibly handling the lexer's output or results.

**Maths:**

Character Classes:

{lower}: Represents the character class for lowercase letters [a-z].
{CAPS}: Represents the character class for uppercase letters [A-Z].
{space}: Represents the character class for space characters [ \t\n].

Regular Expressions:

{lower}: This pattern matches a single lowercase letter. When a lowercase letter is encountered in the input text, the associated action subtracts 32 from its ASCII value, effectively converting it to uppercase.
{CAPS}: This pattern matches a single uppercase letter. When an uppercase letter is encountered, the associated action adds 32 to its ASCII value, converting it to lowercase.
{space}: This pattern matches space characters (including spaces, tabs, and newlines). The associated action echoes (retains) these characters in the output.

Action:

The printf statements within the actions are used to manipulate the characters based on their ASCII values:

{printf("%c", yytext[0]- 32);}: Converts lowercase letters to uppercase by subtracting 32 from their ASCII values and then prints the result as a character.
{printf("%c", yytext[0]+ 32);}: Converts uppercase letters to lowercase by adding 32 to their ASCII values and then prints the result as a character.
{space} ECHO;: Echoes (retains) space characters without any case conversion.
. (dot) pattern: Echoes all other characters (including punctuation and digits) without any case conversion.

**Algorithm:**

1. Start the program.

2. Define the character classes:
   - lowercase: [a-z]
   - uppercase: [A-Z]
   - space: [ \t\n]

3. Initialize a loop to read characters from the input text one at a time:
   - Read the next character from the input text.

4. Check if the character is a lowercase letter:
   - If yes, convert it to uppercase by subtracting 32 from its ASCII value.
   - Print the converted character.

5. Check if the character is an uppercase letter:
   - If yes, convert it to lowercase by adding 32 to its ASCII value.
   - Print the converted character.

6. Check if the character is a space (including spaces, tabs, or newlines):
   - If yes, print the space character as is.

7. For all other characters (including punctuation and digits):
   - Print the character as is.

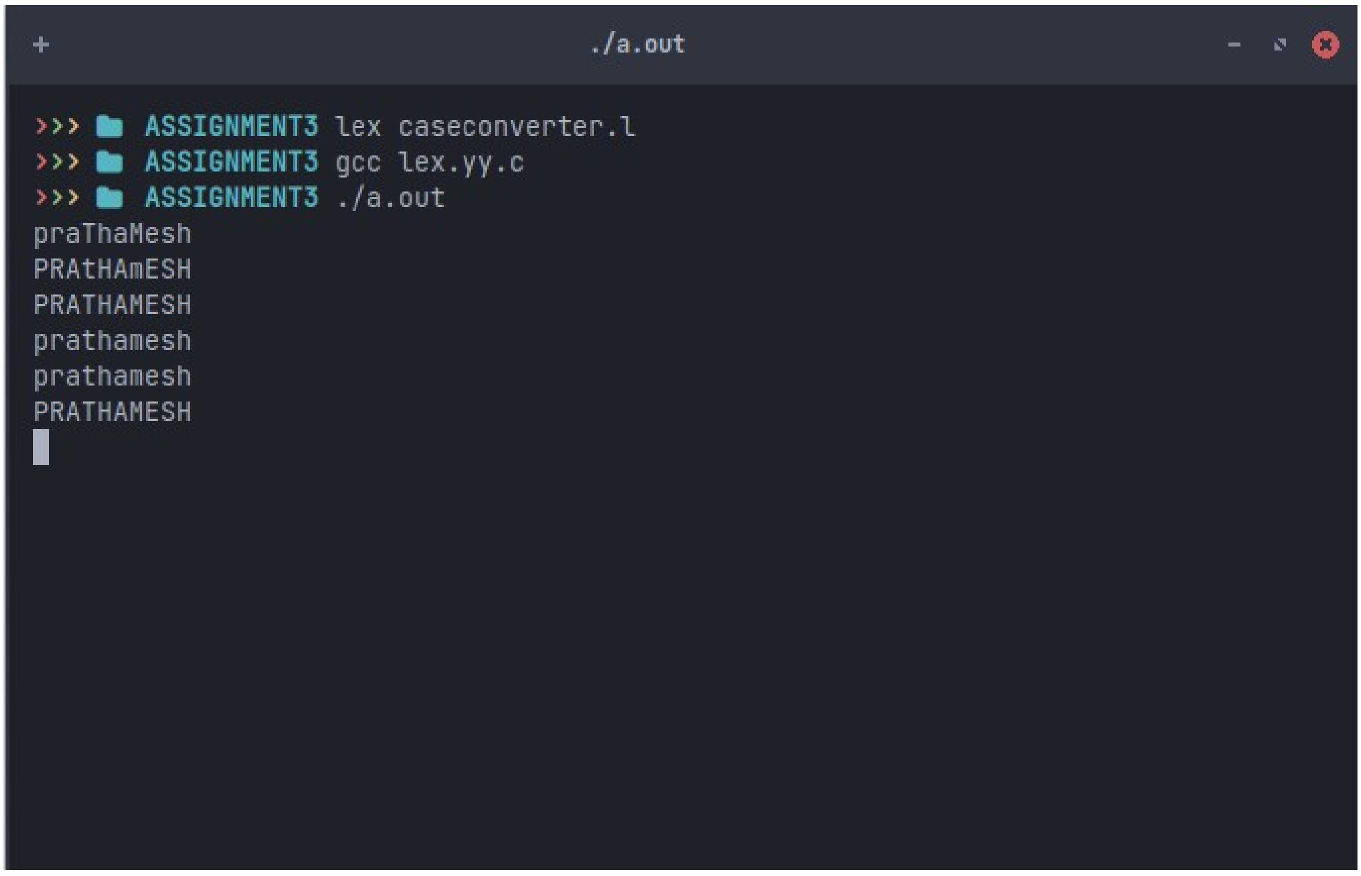
8. Repeat steps 3 to 7 until the end of the input text is reached.

9. End the program.

**Code**

```
lower [a-z]
CAPS  [A-Z]
space    [ \t\n]

%%
{lower} {printf("%c",yytext[0]- 32);}

{CAPS}  {printf("%c",yytext[0]+ 32);}

{space}   ECHO;
.         ECHO;

%%

int yywrap(){return (1);}

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

**Output**



```
./a.out

>>> ASSIGNMENT3 lex caseconverter.l
>>> ASSIGNMENT3 gcc lex.yy.c
>>> ASSIGNMENT3 ./a.out
praThaMesh
PRAtHAmESH
PRATHAMESH
prathamesh
prathamesh
PRATHAMESH
```

## Conclusion

In this assignment, we utilized Lex, a lexical analyzer generator, to develop a program focused on converting the cases of the given input from upper to lower and vise versa by manipulating their ascii values adding and subtracting 32 to perfrom each operation.