COMPILER DESIGN LAB

LAB – 5 INTRODUCTION TO FLEX

ADITYA AGARWAL **220905106** **ROLL NO. 14**

Q1 - Count the number of vowels and consonants in the given input.

CODE:

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

int vowels = 0;
int consonants = 0;
%}

%%
[aAeEiIoOuU]    { vowels++; }    /* Matches vowels */
[b-df-hj-np-tv-zB-DF-HJ-NP-TV-Z]    { consonants++; }    /* Matches consonants */
.|\n    { /* Ignore other characters */ }
%%

int yywrap() {
    return 1;
}

int main() {
    printf("Enter a string: ");
    yylex();

    /* Print the final counts */
    printf("\nResults:\n");
    printf("Number of vowels: %d\n", vowels);
    printf("Number of consonants: %d\n", consonants);

    return 0;
}
```

OUTPUT:

```
CD_LAB_B1@debianpc-02: ~/Desktop/220905106/LAB5/Q1
File  Edit  View  Search  Terminal  Help
CD_LAB_B1@debianpc-02:~/Desktop/220905106/LAB5/Q1$ ./vowels
Enter a string: aditya

Results:
Number of vowels: 3
Number of consonants: 3
CD_LAB_B1@debianpc-02:~/Desktop/220905106/LAB5/Q1$
```

Q2 - Count the number of words, characters, blanks and lines in a given text.

CODE:

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

int words = 0;
int chars = 0;
int blanks = 0;
int lines = 0;
%}

%%

[a-zA-Z0-9]+  { words++; chars += yyleng; }  // Match a word, count characters in the word
[ \t]         { blanks++; chars++; }         // Match spaces and tabs, count as blanks and characters
\n            { lines++; chars++; }          // Match a newline, count lines and newline as a character
.             { chars++; }                   // Match any other character, count it as a character

%%

int yywrap() { return 1; }

int main() {
    printf("Enter the text (Ctrl+D to stop): \n");

    yylex();  // Start lexical analysis
```

```
    // Output the counts after analysis
    printf("Words: %d\n", words);
    printf("Characters: %d\n", chars);
    printf("Blanks: %d\n", blanks);
    printf("Lines: %d\n", lines);

    return 0;
}
```

OUTPUT:

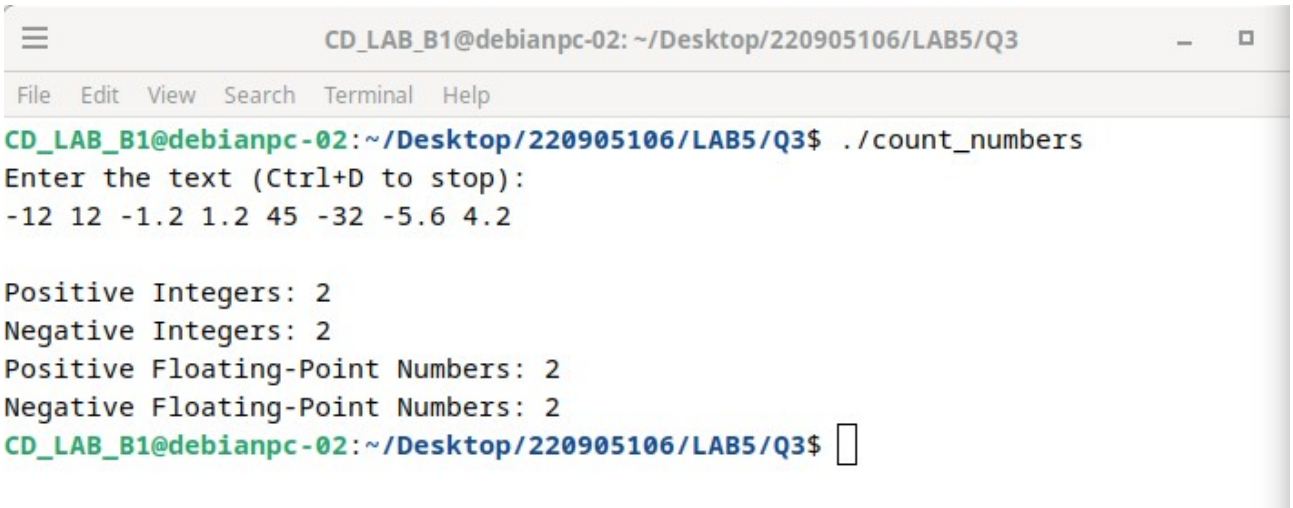
Q3- Find the number of positive integer, negative integer, positive floating positive number and negative floating point number

CODE:

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

int pos_int = 0;
int neg_int = 0;
int pos_float = 0;
int neg_float = 0;
%}

%%

[1-9][0-9]*          { pos_int++; }      // Matches positive integers
-[1-9][0-9]*         { neg_int++; }      // Matches negative integers
[1-9][0-9]*\.[0-9]+   { pos_float++; }    // Matches positive floating-point numbers
```

```
-[1-9][0-9]*\.[0-9]+  { neg_float++; }    // Matches negative floating-point numbers

%%

int yywrap() { return 1; }

int main() {
    printf("Enter the text (Ctrl+D to stop): \n");

    yylex();  // Start lexical analysis

    // Output the counts after analysis
    printf("Positive Integers: %d\n", pos_int);
    printf("Negative Integers: %d\n", neg_int);
    printf("Positive Floating-Point Numbers: %d\n", pos_float);
    printf("Negative Floating-Point Numbers: %d\n", neg_float);

    return 0;
}
```

OUTPUT:



Q4 - Given a input C file, replace all scanf with READ and printf with WRITE statements also find the number of scanf and printf in the file.

CODE:

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

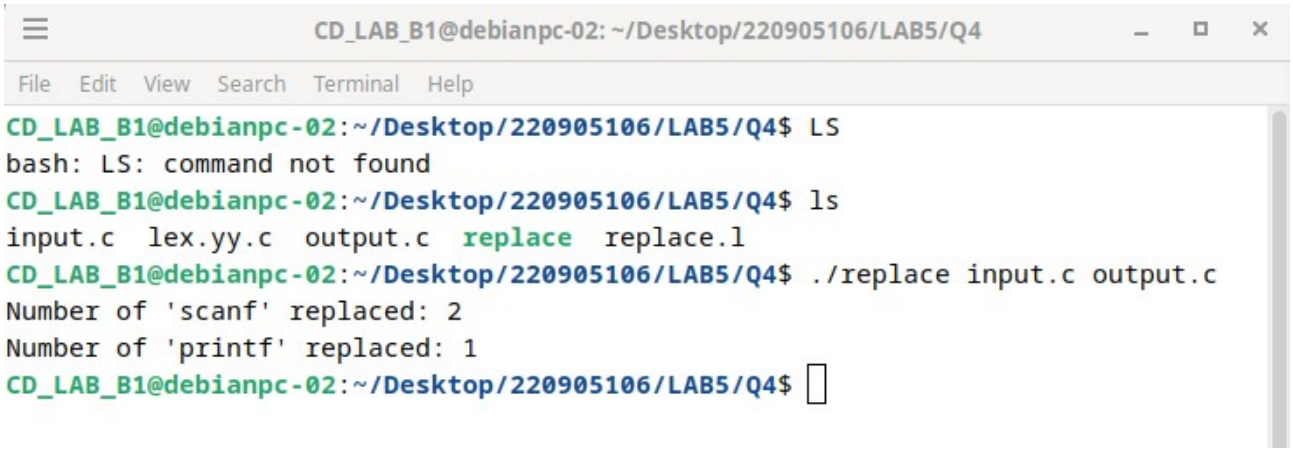
```
int count_scanf = 0;  // Count the number of scanf
int count_printf = 0; // Count the number of printf

// Declare the input and output files globally
FILE *yyin;  // Input file
FILE *yyout; // Output file
%}

%%

scanf\([^\)]*\)   {  // Match 'scanf' with any characters inside parentheses
    count_scanf++;   // Increment the scanf count
    fprintf(yyout, "READ%s", yytext + 5);  // Replace 'scanf' with 'READ'
}

printf\([^\)]*\)   {  // Match 'printf' with any characters inside parentheses
    count_printf++;   // Increment the printf count
    fprintf(yyout, "WRITE%s", yytext + 6);  // Replace 'printf' with 'WRITE'
}

.|\n {  // For any other characters, print them as-is
    fputc(yytext[0], yyout);  // Write the current character to the output file
}

%%

int yywrap() {
    return 1;  // End the scanning when the input is finished
}

int main(int argc, char **argv) {
    if (argc != 3) {
        fprintf(stderr, "Usage: %s <input_file> <output_file>\n", argv[0]);
        return 1;
    }

    // Open the input file for reading
    yyin = fopen(argv[1], "r");
    if (yyin == NULL) {
        perror("Error opening input file");
        return 1;
    }

    // Open the output file for writing
    yyout = fopen(argv[2], "w");
    if (yyout == NULL) {
        perror("Error opening output file");
        fclose(yyin);
```

```
        return 1;
    }

    // Perform lexical analysis
    yylex();

    // Output the counts after the analysis
    printf("Number of 'scanf' replaced: %d\n", count_scanf);
    printf("Number of 'printf' replaced: %d\n", count_printf);

    // Close the files
    fclose(yyin);
    fclose(yyout);

    return 0;
}
```

INPUT.C -

```
#include <stdio.h>

int main() {
    int a;
    scanf("%d", &a);
    printf("The value is: %d", a);
    scanf("%d", &a);
    return 0;
}
```

OUTPUT:



Q5 - That changes a number from decimal to hexadecimal notation.

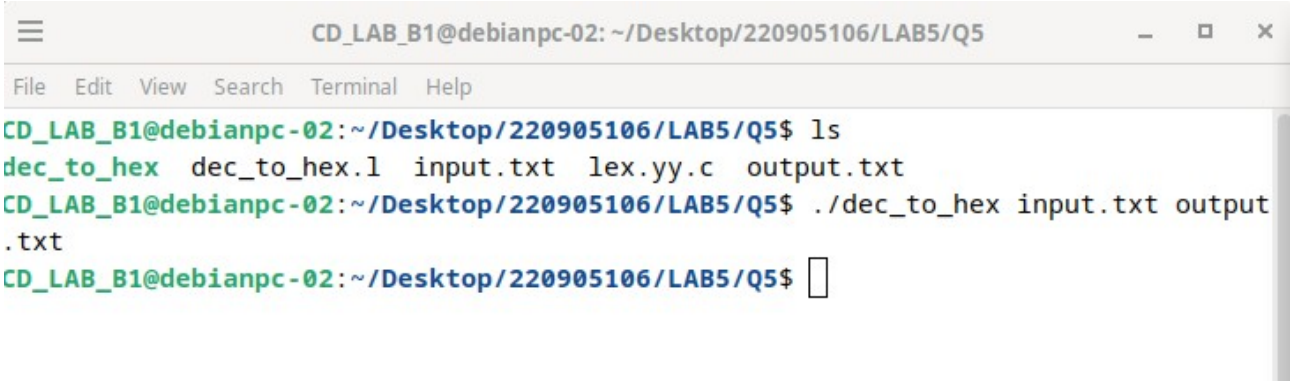
CODE:

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

/* File pointers for input and output */
FILE *yyin, *yyout;
%}

%%
[0-9]+ {
    int num = atoi(yytext);
    fprintf(yyout, "0x%X", num);
}
[ \t\n] ECHO;  /* Echo whitespace characters as they are */
. ECHO;        /* Echo all other characters as they are */
%%

int yywrap(void) {
    return 1;
}

int main(int argc, char *argv[]) {
    /* Check if correct number of arguments */
    if(argc != 3) {
        printf("Usage: %s input-file output-file\n", argv[0]);
        return 1;
    }

    /* Open input file */
    if(!(yyin = fopen(argv[1], "r"))) {
        printf("Cannot open input file\n");
        return 1;
    }

    /* Open output file */
    if(!(yyout = fopen(argv[2], "w"))) {
        printf("Cannot open output file\n");
        return 1;
    }

    /* Run the lexical analyzer */
    yylex();

    /* Close the files */
```

```
    fclose(yyin);
    fclose(yyout);
    return 0;
}
```

INPUT.TXT - The decimal numbers are: 10, 255, 1234, 999.

OUTPUT: The decimal numbers are: 0xA, 0xFF, 0x4D2, 0x3E7.



Q6 - Convert uppercase characters to lowercase characters of C file excluding the characters present in the comment.

CODE:

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

/* File pointers for input and output */
FILE *yyin, *yyout;
int in_comment = 0;  /* Flag to track if we're inside a comment */
int in_string = 0;   /* Flag to track if we're inside a string */
%}

%x COMMENT_SINGLE
%x COMMENT_MULTI
%x STRING

%%
"/*"                { BEGIN(COMMENT_MULTI); fprintf(yyout, "%s", yytext); }
<COMMENT_MULTI>"*/"     { BEGIN(INITIAL); fprintf(yyout, "%s", yytext); }
<COMMENT_MULTI>.|\n     { fprintf(yyout, "%s", yytext); }

"//"                { BEGIN(COMMENT_SINGLE); fprintf(yyout, "%s", yytext); }
<COMMENT_SINGLE>\n      { BEGIN(INITIAL); fprintf(yyout, "%s", yytext); }
<COMMENT_SINGLE>.       { fprintf(yyout, "%s", yytext); }
```

```
\"                    { BEGIN(STRING); fprintf(yyout, "%s", yytext); }
<STRING>\\\"          { fprintf(yyout, "%s", yytext); }  /* Escaped quote in string */
<STRING>\"            { BEGIN(INITIAL); fprintf(yyout, "%s", yytext); }
<STRING>.             { fprintf(yyout, "%s", yytext); }

[A-Z]                 { fprintf(yyout, "%c", yytext[0] + 32); }  /* Convert uppercase to
lowercase */
.|\n                  { fprintf(yyout, "%s", yytext); }  /* Copy everything else as is */

%%

int yywrap(void) {
    return 1;
}

int main(int argc, char *argv[]) {
    if(argc != 3) {
        printf("Usage: %s input-file output-file\n", argv[0]);
        return 1;
    }

    /* Open input file */
    yyin = fopen(argv[1], "r");
    if(yyin == NULL) {
        printf("Cannot open input file %s\n", argv[1]);
        return 1;
    }

    /* Open output file */
    yyout = fopen(argv[2], "w");
    if(yyout == NULL) {
        printf("Cannot open output file %s\n", argv[2]);
        fclose(yyin);
        return 1;
    }

    /* Run the lexical analyzer */
    yylex();

    /* Close the files */
    fclose(yyin);
    fclose(yyout);
    printf("Conversion completed successfully!\n");
    return 0;
}
```

INPUT.C -

```
/* This is a MULTI-LINE Comment
   SHOULD NOT BE CONVERTED */
#include <stdio.h>

// THIS IS A SINGLE LINE COMMENT
int MAIN() {
    printf("THIS STRING SHOULD NOT BE CONVERTED\n");
    int COUNT = 0;
    return 0;
}
```

OUTPUT.C -