LAB 2

1) Write a LEX program to recognize the following

Operators: +, -, *, /, |,

Numbers

newline

Any other character apart from the above should be recognized as mystery character

For each of the above-mentioned matches (classes of lexeme) in your input, the program should print the following: PLUS, MINUS, MUL, DIV, ABS, NUMBER, NEW LINE, MYSTERY CHAR respectively. Your program should also strip of whitespaces.

```
%%
[ \t]+
\n     printf("NEW LINE\n");
[0-9]+ printf("NUMBER\n");
"+"    printf("PLUS\n");
"-"    printf("MINUS\n");
"*"    printf("MUL\n");
"/"    printf("DIV\n");
"|"    printf("ABS\n");
.      printf("MYSTERY CHAR\n");
%%
```

2) Write a LEX program to print the number of words, characters and lines in a given input.

```
%{
    #include <stdio.h>
    int w_count=0,c_count=0,l_count=0;
%}
%%
[A-Za-z]+ { w_count++; c_count+=yyleng; }
\n { c_count++; l_count++; }
. { c_count++; }
%%
int main(int argc, char **argv) {
    yylex();
    printf("Number of characters - %d\n",c_count);
    printf("Number of words - %d\n",w_count);
    printf("Number of lines - %d\n",l_count);
    return 0;
}
```

3) Modify the above LEX program so that a word and its characters are counted only if its length is greater than or equal to 6.

```
%{
    #include <stdio.h>
    int w_count=0,c_count=0,l_count=0;
%}
%%
[A-Za-z]+ { if (strlen(yytext)>=6) { w_count++; } c_count+=yyleng; }
\n { c_count++; l_count++; }
. { c_count++; }
%%
int main(int argc, char **argv) {
    yylex();
    printf("Number of characters - %d\n",c_count);
    printf("Number of words - %d\n",w_count);
    printf("Number of lines - %d\n",l_count);
    return 0;
}
```

4) Write a LEX program to print if the input is an odd number or an even number along with its length. Also, the program should check the correctness of the input (i.e. if the input is one even number and one odd number).

```
%{
    #include<stdlib.h>
    #include<stdio.h>
    int n1;
    int n2;
%}
%%
[0-9]*[0|2|4|6|8] { printf("Even number - %d\n", atoi(yytext)); return
atoi(yytext); }
[0-9]*[1|3|5|7|9] { printf("Odd number - %d\n", atoi(yytext)); return
atoi(yytext); }
%%
int main() {
    printf("Enter an even number - ");
    n1=yylex();
    printf("Enter an odd number - ");
    n2=yylex();
    if ((n1-n2)%2!=0) { printf("Correct Input\n"); }
    else { printf("Incorrect Input\n"); }
    return 1;
}
```

LAB 3

1) Write a LEX program to get a binary input and print whether the given input is a power of two or not.

```
%{
    #include <stdio.h>
    #include <stdlib.h>
%}
%%
[0]*1[0]+ printf("Input is a power of two\n");
[0-1]+    printf("Input is not a power of two\n");
%%
int main() {
    yylex();
    return 1;
}
```

2) Write a LEX program to insert line numbers to a file. For this copy your favourite C program “input.c” to your folder which would be the input to your LEX program.

```
%{
    #include <stdio.h>
    #include <stdlib.h>
    int N=0;
    FILE *output_file;
%}
%%
\n { N++; fprintf(output_file,"%d\n",N); }
.
%%
int main() {
    FILE *input_file;
    input_file=fopen("input.c","r");
    output_file=fopen("output.dat","w");
    yyin=input_file;
    yylex();
    fclose(input_file);
    fclose(output_file);
    return 0;
}
```

3) Write a LEX program to save the contents of an input file excluding comment lines to another file.

```
%{
    #include <stdio.h>
    #include <stdlib.h>
    FILE *output_file;
%}

%%
^\/\/.*
.|\n { fputs(yytext,output_file); }
%%
int main() {
    FILE *input_file=fopen("input.c","r");
    output_file=fopen("output1.dat","w");
    if (!input_file||!output_file) {
        fprintf(stderr, "Error opening files.\n");
        return 1;
    }
    yyin=input_file;
    yylex();
    fclose(input_file);
    fclose(output_file);
    return 0;
}
```

4) Write a LEX program that would take a BITS student's roll number as input and prints the details of the student based on that. You are expected to write regular expressions that would synthesize information like, year of joining, specialization, PS/Thesis, Registration index, Campus (U) etc. from the given roll number. If the given input does not abide by the Roll number format, print some error message.

```
%{
    #include <stdio.h>
    #include <stdlib.h>
%}
%%
^[0-9]{4}A[1|2|3|4|7|9|A](PS|TS)[0-9]{4}[P|H|G|U] { printf("Year of joining -  %c%c%c%c\n", yytext[0], yytext[1], yytext[2],
yytext[3]);
                                                    switch(yytext[5]) {
                                                        case '1':
                                                            printf("Discipline - Chemical Engineering\n");
                                                            break;
                                                        case '2':
                                                            printf("Discipline - Civil Engineering\n");
                                                            break;
                                                        case '3':
                                                            printf("Discipline - Electrical and Electronics Engineering\n");
                                                            break;
                                                        case '4':
                                                            printf("Discipline - Mechanical Engineering\n");
                                                            break;
                                                        case '7':
                                                            printf("Discipline - Computer Science Engineering\n");
                                                            break;
                                                        case '9':
                                                            printf("Discipline - Biotechnological Engineering\n");
                                                            break;
```

```
            case 'A':
                printf("Discipline - Electrical and Communications
Engineering\n");

                break;
            default:
                break;
        }
        switch(yytext[6]) {
            case 'P':
                printf("Internship Mode - Practice School\n");
                break;
            case 'T':
                printf("Internship Mode - Thesis\n");
                break;
            default:
                break;
        }
        printf("Roll Number - %c%c%c%c\n",
yytext[8],yytext[9],yytext[10],yytext[11]);

        switch(yytext[12]) {
            case 'P':
                printf("Campus Location - Pilani\n");
                break;
            case 'U':
                printf("Campus Location - Dubai\n");
                break;
            case 'G':
                printf("Campus Location - Goa\n");
                break;
            case 'H':
                printf("Campus Location - Hyderabad\n");
                break;
```

```
                default:
                    break;
            }
        }
.* { printf("Invalid BITS ID Input"); }
%%
int main() {
        yylex();
        return 1;
}
```

LAB 4

1) **Understand the working of LEX and YACC using a simple calculator application. Your calculator should provide arithmetic operators + and - that can add or subtract integers respectively (once). Start off with a grammar (not perfect).**
   Program → E \n
   E → E + E | E - E | int
   The problems with the above grammar – ambiguity. Associativity is not considered. Understand the conflicts in bottom-up parsing and YACC's default action when there is a conflict.

   Lexer:

```
%{
    #include <stdlib.h>
    void yyerror(char *);
    #include "y.tab.h"
    extern int yylval;
%}
%%
[0-9]+ { yylval=atoi(yytext); return INTEGER; }
"+" return PLUS;
"-" return MINUS;
"\n" return NL;
[ \t]
. printf("Invalid character\n");
%%
int yywrap(void) { return 1; }
```

   Parser:

```
%{
    #include <stdio.h>
    #include <stdlib.h>
    int yylex(void);
    void yyerror(char *);
%}
%token INTEGER PLUS MINUS NL
%%
program:
    expr NL { printf("%d\n", $1); exit(0);}
    ;
expr:
    INTEGER { $$ = $1; }
    | expr PLUS expr { $$ = $1 + $3; }
    | expr MINUS expr { $$ = $1 - $3; }
    ;
%%
void yyerror(char *s) {
fprintf(stderr, "%s\n", s);
}
int main() { yyparse(); return 0; }
```

2) **Modify your program to take care of associativity of the operators using directives in YACC. + and – are left associative. Also modify**

the grammar so the calculator application runs until the user quits using <CTRL + D>.

Lexer:

```
%{
    #include <stdlib.h>
    void yyerror(char *);
    #include "y.tab.h"
%}
%%
[0-9]+ { yylval = atoi(yytext); return INTEGER; }
"+" return PLUS;
"-" return MINUS;
"\n" return NL;
[ \t]
. printf("Invalid character\n");
%%
int yywrap(void) { return 1; }
```

Parser:

```
%{
    #include <stdio.h>
    #include<stdlib.h>
    int yylex(void);
    void yyerror(char *);
%}
%token INTEGER PLUS MINUS NL
%left PLUS MINUS
%%
program:
    program expr NL { printf("%d\n", $2); }
    |
    ;
expr:
    INTEGER { $$ = $1; }
    | expr PLUS expr { $$ = $1 + $3; }
    | expr MINUS expr { $$ = $1 - $3; }
    ;
%%
void yyerror(char *s) { printf("%s\n", s); }
int main() { yyparse(); return 0; }
```

3) **Extend the calculator to incorporate some new functionality. New features include arithmetic operators * and / that can multiply and divide integers respectively. Parentheses may be used to over-ride operator precedence. Note * and / operators have higher precedence over + and – operators. Also note that * and / are left associative. Ensure this using directive in YACC.**

Lexer:

```
%{
    #include <stdlib.h>
    void yyerror(char *);
    #include "y.tab.h"
%}
```

```
%%
[0-9]+ { yylval = atoi(yytext); return INTEGER; }
"+" return PLUS;
"-" return MINUS;
"*" return MUL;
"/" return DIV;
"\n" return NL;
"(" return LP;
")" return RP;
[ \t]
. printf("Invalid character\n");
%%
int yywrap(void) { return 1; }
```

Parser:

```
%{
    #include <stdio.h>
    #include<stdlib.h>
    int yylex(void);
    void yyerror(char *);
%}
%token INTEGER PLUS MINUS NL MUL DIV LP RP
%left PLUS MINUS
%left MUL DIV
%%
program:
    program expr NL { printf("%d\n", $2);}
    |
    ;
expr:
    INTEGER { $$ = $1; }
    | expr PLUS expr { $$ = $1 + $3; }
    | expr MINUS expr { $$ = $1 - $3; }
    | expr MUL expr { $$ = $1 * $3; }
    | expr DIV expr { $$ = $1 / $3; }
    | LP expr RP { $$ = $2;}
    ;
%%
void yyerror(char *s) { printf("%s\n", s); }
int main() { yyparse(); return 0; }
```

4) Modify the grammar to allow single-character variables to be specified in assignment statements. The following illustrates sample input and calculator output:

```
user: 3 * (4 + 5)
calc: 27
user: x = 3 * (4 + 5)
user: y = 5
user: x
calc: 27
user: y
calc: 5
user: x + 2*y
calc: 37
```

Lexer:

```
%{
    #include <stdlib.h>
    void yyerror(char *);
    #include "y.tab.h"
%}
%%
[0-9]+ { yylval=atoi(yytext); return INTEGER; }
[a-z] { yylval=*yytext-'a'; return VAR; }
"+" return PLUS;
"-" return MINUS;
"*" return MUL;
"/" return DIV;
"\n" return NL;
"(" return LP;
")" return RP;
"=" return EQ;
[ \t]
. printf("Invalid character\n");
%%
int yywrap(void) { return 1; }
```

Parser:

```
%{
    #include <stdio.h>
    int yylex(void);
    void yyerror(char *);
    int sym[26];
%}
%token INTEGER PLUS MINUS NL MUL DIV LP RP VAR EQ
%left PLUS MINUS
%left MUL DIV
%%
program:
    program stmt NL
    |
    ;
stmt:
    expr {printf("%d\n", $1);}
    | VAR EQ expr { sym[$1] = $3; }
    ;
expr:
    INTEGER { $$ = $1; }
    | VAR { $$ = sym[$1]; }
    | expr PLUS expr { $$ = $1 + $3; }
    | expr MINUS expr { $$ = $1 - $3; }
    | expr MUL expr { $$ = $1 * $3; }
    | expr DIV expr { $$ = $1 / $3; }
    | LP expr RP { $$ = $2;}
    ;
%%
void yyerror(char *s) { printf("%s\n", s); }
int main() { yyparse(); return 0; }
```

LAB 5

1) Write the LEX and YACC source to recognize the following:

[a] General Template:
The template for the C program is

```
#include<stdio.h>
int main()
{
}
```

PRM → HEADER INT MAIN LB RB LCB RCB

Lexer:

```
%option yylineno
%{
    #include <stdio.h>
    #include "y.tab.h"
    #include <math.h>
%}
%%
"#include<stdio.h>" { return HEADER; }
"int" { return INT; }
"main" { return MAIN; }
"(" { return LB; }
")" { return RB; }
"{" { return LCB; }
"}" { return RCB; }
[\n\t ]+
. { printf("Unmatched character %s in line number %d of the
input\n",yytext,yylineno); }
%%
int yywrap(void) { return 1; }
```

Parser:

```
%{
    #include<stdio.h>
    int yylex(void);
    int yyerror(const char *s);
%}
%token HEADER INT MAIN LB RB LCB RCB
%start prm
%%
prm: HEADER INT MAIN LB RB LCB RCB { printf("Parsing Successful\n"); }
%%
int main() { yyparse(); return 0; }
int yyerror(const char *msg) {
    extern int yylineno;
    printf("Parsing Failed due to %s\n",msg);
    return 0;
}
```

[b] Declaration statements:
Allow declaration statements inside the program body. Integer variables separated by comma can be declared inside the program body. A program can have multiple declaration statements. Variables are sequences of lower-case alphabets. Declarations statements are ended by a semicolon.

int a, b;

PRM → HEADER INT MAIN LB RB LCB BODY RCB
BODY → DECL_STMTS
DECL_STMTS → DECL_STMT DECL_STMTS | DECL_STMT
DECL_STMT → INT VAR_LIST SC
VAR_LIST → VAR COMA VAR_LIST | VAR

Lexer:

```
%option yylineno
%{
    #include<stdio.h>
    #include"y.tab.h"
    #include<math.h>
%}
%%
"#include<stdio.h>" { return HEADER; }
"int" { return INT; }
"main" { return MAIN; }
"(" { return LB; }
")" { return RB; }
"{" { return LCB; }
"}" { return RCB; }
"," { return COMMA; }
";" { return SC; }
[a-z]+ { return VAR; }
[\n\t ]+
. { printf("Unexpected character sequence %s in line no
%d\n",yytext,yylineno); exit(0); }
%%
int yywrap(void) { return 1; }
```

Parser:

```
%{
    #include<stdio.h>
    int yylex(void);
    int yyerror(const char *s);
%}
%token HEADER INT MAIN LB RB LCB RCB SC COMMA VAR
%start prm
%%
prm: HEADER INT MAIN LB RB LCB BODY RCB {printf("Parsing Successful\n");}
    ;
BODY: DECL_STMTS
    ;
DECL_STMTS: DECL_STMT DECL_STMTS
    | DECL_STMT
    ;
DECL_STMT: INT VAR_LIST SC
    ;
```

```
VAR_LIST: VAR COMMA VAR_LIST
    | VAR
    ;
%%
int main() {
    yyparse();
    return 0;
}
int yyerror(const char *msg) {
    extern int yylineno;
    printf("Parsing Failed due to %s in line no %d\n",msg,yylineno);
    return 0;
}
```

[c] OPERATORS & PROGRAM STATEMENTS:
Allow declaration statements to be followed by program statements inside the program body. Program statements are ended by a semicolon. Program statements can be arithmetic expressions involving +-*/ operators.

a = b + c; / a = (b*c)

PRM → HEADER INT MAIN LB RB LCB BODY RCB
BODY → DECL_STMTS PROG_STMTS
DECL_STMTS → DECL_STMT DECL_STMTS | DECL_STMT
PROG_STMTS → PROG_STMT PROG_STMTS | PROG_STMT
DECL_STMT → INT VAR_LIST SC
VAR_LIST → VAR COMA VAR_LIST | VAR
PROG_STMT → VAR EQ A_EXPN SC
A_EXPN → A_EXPN OP A_EXPN | LB A_EXPN RB | VAR

Lexer:

```
%option yylineno
%{
    #include<stdio.h>
    #include"y.tab.h"
    #include<math.h>
%}
%%
"#include<stdio.h>" { return HEADER; }
"int" { return INT; }
"main" { return MAIN; }
"(" { return LB; }
")" { return RB; }
"{" { return LCB; }
"}" { return RCB; }
"," { return COMMA; }
";" { return SC; }
[\+\-\*\/] { return OP; }
"=" { return EQ; }
[a-z]+ { return VAR; }
[\n\t ]+
. { printf("Unexpected character sequence %s in line no
%d\n",yytext,yylineno); exit(0); }
%%
int yywrap(void) { return 1; }
```

Parser:

```
%{
    #include<stdio.h>
    int yylex(void);
    int yyerror(const char *s);
%}
%token HEADER INT MAIN LB RB LCB RCB SC COMMA VAR EQ OP
%start prm
%%
prm: HEADER INT MAIN LB RB LCB BODY RCB {printf("Parsing Successful\n");}
    ;
BODY: DECL_STMTS PROG_STMTS
    ;
DECL_STMTS: DECL_STMT DECL_STMTS
    | DECL_STMT
    ;
DECL_STMT: INT VAR_LIST SC
    ;
VAR_LIST: VAR COMMA VAR_LIST
    | VAR
    ;
PROG_STMTS: PROG_STMT PROG_STMTS
    | PROG_STMT
    ;
PROG_STMT: VAR EQ A_EXPN SC
    ;
A_EXPN: A_EXPN OP A_EXPN
    | LB A_EXPN RB
    | VAR
    ;
%%
int main() {
    yyparse();
    return 0;
}
int yyerror(const char *msg) {
    extern int yylineno;
    printf("Parsing Failed due to %s in line no %d\n",msg, yylineno);
    return 0;
}
```

[d] Modify your LEX program to incorporate the following changes

- **As per the current set up, the programmer is supposed to use only lower-case alphabets in variable names in their C program. Modify your lex program so as to let the programmer have uppercase letters A to Z together with digits 0 to 9 and underscore character in variable names. Ensure that a variable name always begin with a character.**
- **Terminate your program with an error message if in case the programmer uses keywords if, while, do, and for as variable names. Note that its permitted to have variable names beginning with keywords (ifvar, donut etc.) (hint: rely on conflict resolution rules).**
- **Add provision to declare variables of type float, double and char.**

Lexer:

```
%option yylineno
%{
    #include<stdio.h>
    #include"y.tab.h"
    #include<stdlib.h>
%}
%%
"#include<stdio.h>" { return HEADER; }
"int" { return INT; }
"float" { return FLOAT; }
"double" { return DOUBLE; }
"char" { return CHAR; }
"if" { printf("Usage of keyword if  as var name is prohibited \n");
exit(0); }
"while" { printf("Usage of keyword while  as var name is prohibited \n");
exit(0); }
"do" { printf("Usage of keyword do as var name is prohibited \n");
exit(0); }
"for" { printf("Usage of keyword for as var name is prohibited \n");
exit(0); }
"main"      { return MAIN; }
"(" { return LB; }
")" { return RB; }
"{" { return LCB; }
"}" { return RCB; }
"," { return COMMA; }
";" { return SC; }
[\+\-\*\/] { return OP; }
"=" { return EQ; }
[a-z][a-zA-Z0-9]+ { return VAR; }
[\n\t ]+
. { printf("Unexpected character sequence %s in line no
%d\n",yytext,yylineno); exit(0); }
%%
int yywrap(void) { return 1; }
```

Parser:

```
%{
    #include<stdio.h>
    int yylex(void);
    int yyerror(const char *s);
%}
%token HEADER INT MAIN LB RB LCB RCB SC COMMA VAR EQ OP FLOAT DOUBLE CHAR
%start prm
%%
prm: HEADER INT MAIN LB RB LCB BODY RCB { printf("Parsing Successful\n");
}
    ;
BODY: DECL_STMTS PROG_STMTS
    ;
DECL_STMTS: DECL_STMT DECL_STMTS
    | DECL_STMT
    ;
DECL_STMT: INT VAR_LIST SC
    | FLOAT VAR_LIST SC
    | DOUBLE VAR_LIST SC
    | CHAR VAR_LIST SC
    ;
VAR_LIST: VAR COMMA VAR_LIST
    | VAR
    ;
PROG_STMTS: PROG_STMT PROG_STMTS
    | PROG_STMT
    ;
PROG_STMT: VAR EQ A_EXPN SC
    ;
A_EXPN: A_EXPN OP A_EXPN
    | LB A_EXPN RB
    | VAR
    ;
%%
int main() { yyparse(); return 0; }
int yyerror(const char *msg) {
    extern int yylineno;
    printf("Parsing Failed due to %s in line no %d\n",msg,yylineno);
    return 0;
}
```