Practical 1

```
%{
#include <stdio.h>
#include <stdlib.h>
void yyerror(const char *s);
%}
digit               [0-9]
letter              [a-zA-Z]
identifier          {letter}({letter}|{digit}|_)
keyword
"int"|"float"|"char"|"void"|"if"|"else"|"for"|"whil
e"|"return"
operator
"+"|"-"|"*"|"/"|"="|"=="|"!="|"<"|">"|"<="|">="
separator            ";"|","|"\("|\)|"\{"|"\}"
comment
"//".*|"/\*[^*]*\*+([^/*][^*]*\*+/)?"
whitespace           [ \t]+
newline              \n
string               \"([^\\\"\n]|\\.)*\"
%%
{identifier}       { printf("ID: %s\n", yytext); }
{keyword}           { printf("KEYWORD:
%s\n", yytext); }
{digit}            { printf("NUMBER: %s\n",
yytext); }
{operator}          { printf("OPERATOR:
%s\n", yytext); }
{separator}         { printf("SEPARATOR:
%s\n", yytext); }
{comment}           { /* Ignore comments */ }
{whitespace}        { /* Ignore whitespace */ }
{newline}          { printf("NEW LINE\n"); }
{string}           { printf("STRING: %s\n",
yytext); }
.                  { printf("ERROR:
Unrecognized character: %s\n", yytext); }
%%
void yyerror(const char *s) {
    fprintf(stderr, "Error: %s\n", s);
}
int main() {
    printf("Enter your C code (end with
Ctrl+D):\n");
    yylex(); // Call the lexer
    return 0;
}
```

give commands

1.flex lexer.l
2.gcc lex.yy.c -o lexer -lfl
3. ./lexer

output :

```
Enter your C code (end with Ctrl+D):
int a = 10;
float b = 20.5;
if (a < b) {
    printf("a is less than b");
}
// This is a comment
KEYWORD: int
ERROR: Unrecognized character: a
OPERATOR: =
NUMBER: 1
NUMBER: 0
SEPARATOR: ;
NEW LINE
KEYWORD: float
ERROR: Unrecognized character: b
OPERATOR: =
NUMBER: 2
NUMBER: 0
ERROR: Unrecognized character: .
NUMBER: 5
SEPARATOR: ;
NEW LINE
ID: if
SEPARATOR: (
ERROR: Unrecognized character: a
OPERATOR: <
ERROR: Unrecognized character: b
SEPARATOR: )
SEPARATOR: {
NEW LINE
ID: pr
KEYWORD: int
ERROR: Unrecognized character: f
SEPARATOR: (
STRING: "a is less than b"
SEPARATOR: )
SEPARATOR: ;
NEW LINE
SEPARATOR: }
```

Practical 2

```
%{
#include <stdio.h>
#include <stdlib.h>
%}
Binary    [0-1]+
Oct       [0-7]+
Dec       [0-9]+
Hex       [0-9A-Fa-f]+
Float     [-+]?[0-9]*\.[0-9]+
Exponent
[-+]?[0-9]*\.?[0-9]+([eE][-+]?[0-9]+)?
%%
{Binary}   { printf("This is a binary number:
%s\n", yytext); }
{Oct}      { printf("This is an octal number:
%s\n", yytext); }
{Dec}      { printf("This is a decimal number:
%s\n", yytext); }
{Hex}      { printf("This is a hexadecimal
number: %s\n", yytext); }
{Float}    { printf("This is a float number:
%s\n", yytext); }
{Exponent} { printf("This is an exponent
number: %s\n", yytext); }
.|\n       { }

%%

int main()
{
   printf("Enter numbers (end with Ctrl+D or
Ctrl+Z):\n");
   yylex();
   return 0;
}
```

give commands

1.flex lexer.l
2.gcc lex.yy.c -o lexer -lfl
3. ./lexer

output

```
Enter numbers (end with Ctrl+D or Ctrl+Z):

1235
This is an octal number: 1235
1010
This is a binary number: 1010
10.32
This is a float number: 10.32
ab1
This is a hexadecimal number: ab1
```

Practical 3

Lexer.l

```
%{
#include "parser.tab.h"  // Ensure this is included to define tokens
#include <stdio.h>
#include <stdlib.h>
extern int yylval; // Declare yylval for the parser
%}
%%
[0-9]+ {
    yylval = atoi(yytext);
    return NUMBER;
}
[\t]+ ; // Ignore whitespace
\n { return 0; } // Handle new lines
. { return yytext[0]; } // Return single characters
%%
int yywrap() {
    return 1;
}
```

Parser.y

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

int yylex(); // Declare the yylex function
int flag = 0; // To track if the expression is valid
void yyerror(const char *s);
%}
%token NUMBER
%left '+' '-'
%left '*' '/' '%'
%left '(' ')'
%%
// Grammar rules
ArithmeticExpression: E {
    printf("\nResult = %d\n", $1);
    return 0;
}
;
E: E '+' E { $$ = $1 + $3; }
 | E '-' E { $$ = $1 - $3; }
 | E '*' E { $$ = $1 * $3; }
 | E '/' E { $$ = $1 / $3; }
 | E '%' E { $$ = $1 % $3; }
 | '(' E ')' { $$ = $2; }
 | NUMBER { $$ = $1; }
 ;
%%
// Driver code
int main() {
    printf("Enter any arithmetic expression:\n");
    yyparse();
    if (flag == 0)
        printf("\nEntered arithmetic expression is Valid\n\n");
    return 0;
}
void yyerror(const char *s) {
    printf("\nEntered arithmetic expression is Invalid: %s\n\n", s);
    flag = 1;
}
```

enter command :

1. sudo apt-get install bison flex
2. bison -d parser.y
3. lex lexer.l
4. gcc -o calc parser.tab.c lex.yy.c -lfl
5. ./calc

Output :

```
Enter any arithmetic expression:
2+5-1*4/3

Result = 6

Entered arithmetic expression is Valid
```

Practical 4

create lexer.l

```
%{
#include "parser.tab.h"
int yylex();
%}
%%
"0"      { return ZERO; }
"1"      { return ONE; }
[\n]     { return NL; }
[ \t]+   ;
.        { }
%%
int yywrap() {
    return 1;
}
```

create parser.y

```
%{
#include <stdio.h>
#include <stdlib.h>
int flag = 0;
void yyerror(const char *s);
%}

%token ONE ZERO NL
%%
// Grammar rules
str1:
    str2 n1 { }
;
str2:
    ZERO str2 ONE { }
  | ZERO ONE { }
;
n1:
    NL {
        if (flag == 0) {
            printf("\nThe string is accepted.\n");
        }
        return 0;
    }
;

// Catch-all for any invalid patterns
%%
int main() {
    printf("Enter string (any combination of 0
and 1):\n");
    yyparse();
    if (flag == 0) {
        printf("\nThe string is valid for
L=[0^n1^n].\n");
    }
    return 0;
}
void yyerror(const char *s) {
    printf("\nEntered arithmetic is invalid for
L=[0^n1^n]: %s\n\n", s);
    flag = 1;
}
```

commands on terminal

```
 1)flex lexer.l
 2)bison -d parser.y
 3)gcc lex.yy.c parser.tab.c -o parser -lfl
 4)./parser
```

output:

```
Enter string (any combination of 0 and 1):
0101

Entered arithmetic is invalid for L=[0^n1^n]:
syntax error

Enter string (any combination of 0 and 1):
0011

The string is accepted.

The string is valid for L=[0^n1^n].
```

Practical 5

create lexer.l

```
%{
#include <stdio.h>
#include <string.h>
#include "y.tab.h"  // Include Yacc header
for token definitions

FILE *fpOut;
%}

%%
[a-zA-Z_][a-zA-Z0-9_]* {  // Variable names
    strcpy(yylval.vname, yytext);
    return NAME;
}
[0-9]+          {  // Numbers
    yylval.num = atoi(yytext);
    return NUMBER;
}
[ \t]+          ;  // Ignore whitespace
\n              { return '\n'; }
"="             { return '='; }
"+"             { return '+'; }
"-"             { return '-'; }
"*"             { return '*'; }
"/"             { return '/'; }
.               { return yytext[0]; }  // Return
single characters
%%

// Declare the input file
FILE *yyin;

int main() {
    FILE *fpInput;
    fpInput = fopen("input.txt", "r");
    fpOut = fopen("output.asm", "w");
    yyin = fpInput;
    yyparse();
    fclose(fpInput);
    fclose(fpOut);
    return 0;
}
```

create parser.y

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

extern int yylex();  // Declare yylex
void yyerror(const char *s); // Declare
yyerror
FILE *fpOut;  // Declare fpOut globally

%}

%union {
    char vname[32];  // Variable names
    int num;         // Numeric values
}

%token <vname> NAME
%token <num> NUMBER
%type <vname> expression

%left '+' '-'
%left '*' '/'

%%
input: line '\n' input
     | '\n' input
     | ;

line: NAME '=' expression {
    fprintf(fpOut, "MOV %s, AX\n", $1);
}
;

expression: expression '+' expression {
    fprintf(fpOut, "ADD AX, %s\n", $3);
}
| expression '-' expression {
    fprintf(fpOut, "SUB AX, %s\n", $3);
```

```
}
| expression '*' expression {
    fprintf(fpOut, "MUL AX, %s\n", $3);
}
| expression '/' expression {
    fprintf(fpOut, "DIV AX, %s\n", $3);
}
| NAME {
    fprintf(fpOut, "MOV AX, %s\n", $1);
}
| NUMBER {
    fprintf(fpOut, "MOV AX, %d\n", $1);
}
;

%%

// Error handling function
void yyerror(const char *s) {
    fprintf(stderr, "Error: %s\n", s);
}
```

create input.txt in same folder

```
a = 10
b = 5
c = a + b
d = c * 3
e = d - 2
```

commands for terminal

1.yacc -d your_yacc_file.y
2.lex your_lex_file.l
3.gcc lex.yy.c y.tab.c -o your_compiler -lfl
4. ./your_compiler
5. ls -l output.asm # Check permissions
6. sudo apt install gedit
7. gedit output.asm

Practical 6

create flexer.l

```
 %{
#include "parser.tab.h"
%}
%%
[01]      { return yytext[0] - '0'; } // Return 0 or 1 as token
[\n]      { return NL; }
[ \t]     ;  // Ignore spaces and tabs
.         { /* Ignore any other character */ }
%%
```

create parser.y

```
%{
#include <stdio.h>
void yyerror(const char *s);
int yylex();
%}
%token ZERO ONE NL
%start q0
%%
q0: ZERO q0 { /* Stay in q0 for 0 */ }
  | ONE q1 { /* Transition to q1 for 1 */ }
  | ZERO q0 ZERO { /* Handle consecutive 0s */ }
  | ZERO q0 ONE { /* Handle 0 followed by 1 */ }
  | ONE q1 ZERO { /* Handle 1 followed by 0 */ }
  | ONE q1 ONE { /* Handle consecutive 1s */ }
  | NL { printf("Number is divisible by 3\n"); }
  ;
q1: ZERO q1 { /* Transition back to q0 for 0 */ }
  | ONE q2 { /* Transition to q2 for 1 */ }
  | ZERO q1 ZERO { /* Handle consecutive 0s */ }
  | ZERO q1 ONE { /* Handle 0 followed by 1 */ }
  | ONE q2 ZERO { /* Handle 1 followed by 0 */ }
  | ONE q2 ONE { /* Handle consecutive 1s */ }
  | NL { printf("Number is not divisible by 3, remainder is 1\n"); }
  ;

q2: ZERO q2 { /* Stay in q2 for 0 */ }
  | ONE q0 { /* Transition back to q0 for 1 */ }
  | ZERO q2 ZERO { /* Handle consecutive 0s */ }
  | ZERO q2 ONE { /* Handle 0 followed by 1 */ }
  | ONE q0 ZERO { /* Handle 1 followed by 0 */ }
  | ONE q0 ONE { /* Handle consecutive 1s */ }
  | NL { printf("Number is not divisible by 3, remainder is 2\n"); }
  ;
%%

void yyerror(const char *s) {
    fprintf(stderr, "Error: %s\n", s);
}

int main() {
    printf("\nEnter a binary number to check its divisibility by 3:\n");
    yyparse();
    return 0;
}
```

Commands

```
 1)flex lexer.l
 2)bison -d parser.y
 3)gcc -o check_divisibility parser.tab.c lex.yy.c -lfl
 4)./check_divisibility
```