1. Write a LEX code to check date is valid or not Coding (text):

```
%option noyywrap
%{
#include<stdio.h> #include<stdlib.h> int check=0;
void check_date(){ if(check){ printf("Valid date"); }
}
%}
%%
[1-31]+[/]+[JAN|MAR|MAY|JUL|AUG|OCT|DEC]+[/]+[0-2024] {printf("It is a valid date"); check=0;}
[1-30]+[/]+[APR|JUN|NOV|SEP]+[/]+[0-2024] {printf("It is a valid date"); check=0;}
[1-29]+[/]+[FEB]+[/]+[0-2024] {printf("It is a valid date"); check=0;}
.|\n {check=1;}
%%
int main(){
yylex();
check_date();
return 0;
}
```

2. Write a LEX code to insert the line number in a file.

Coding (text):

```
%option noyywrap
%{
#include <stdio.h>
int line_num = 1; %}
%%
\n { printf("\n%d ", line_num++); } .|\n { printf("%s", yytext); }
%%
int main() {
yyin = fopen("input.txt", "r"); if (!yyin) {
perror("Error opening input file");
return 1; }
printf("%d ", line_num++); yylex();
fclose(yyin);
return 0; }
```

3. Write a LEX program to print the longest string in a file.

Coding (text):

```
%option noyywrap
%{
#include <stdio.h> #include <string.h> char longest[1000] = ""; char current[1000];
%}
%% [a-zA-Z]+ {
strcpy(current, yytext);
if (strlen(current) > strlen(longest))
strcpy(longest, current);
}
.|\n {} %%
int main() {
yyin = fopen("input.txt", "r"); if (!yyin) {
perror("Error opening input file");
return 1; }
yylex();
printf("The longest string in the file: %s\n", longest); fclose(yyin);
return 0;
}
```

Q1. write a yacc program to implement strings of {a^nb,n>=1}
code:
lex code:

```
%{
#include "y.tab.h"
%}
%%
"a" { return A; } "b" { return B; } [\t]+;
['\n'] {return '\n';}
. {return yytext[0];} %%
int yywrap() { return 1;
}
```

yacc code:

```
%{
#include<stdio.h> #include<stdlib.h>
int yylex();
int yyerror(char *msg);
%}
%token A B;
%%
E:S'\n' {printf("String Accepted\n");
exit(0);
} S:A S
|B ;
%%
int main(){
printf("Enter the string: \n"); yyparse();
Compiler Design Lab
}
int yyerror(char *msg){
printf("Invalid String\n");
exit(0); }
```

Q1: YACC Program to Convert an Expression into Prefix Form and Evaluate It
Lex Code:
lex
Copy code

```
%{
#include "y.tab.h"
%}
%%
[0-9]+ { yylval = atoi(yytext); return NUM; }
"*" { return MUL; }
"/" { return DIV; }
"+" { return ADD; }
"-" { return SUB; }
"(" { return LP; }
")" { return RP; }
\n { return 0; }
[ \t]+; /* Ignore whitespace */
. { return yytext[0]; }
%%
int yywrap() { return 1; }
```

YACC Code:
yacc
Copy code
```
%{
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
int yyparse();
int yylex();
int yyerror(const char *msg);
void to_prefix(char op, char *left, char *right, char *result);
%}
%token NUM ADD SUB MUL DIV LP RP
%left ADD SUB
%left MUL DIV
%%
E : E ADD E { to_prefix('+', $1, $3, $$); }
 | E SUB E { to_prefix('-', $1, $3, $$); }
 | E MUL E { to_prefix('*', $1, $3, $$); }
 | E DIV E { to_prefix('/', $1, $3, $$); }
 | LP E RP { $$ = $2; }
 | NUM    { sprintf($$, "%d", $1); }
 ;
%%
char prefix[100];
void to_prefix(char op, char *left, char *right, char *result) {
   sprintf(result, "%c %s %s", op, left, right);
   strcpy(prefix, result);
}
int yyerror(const char *msg) {
   fprintf(stderr, "Error: %s\n", msg);
   return 0;
}
int main() {
   printf("Enter an arithmetic expression: ");
   yyparse();
   printf("Prefix Form: %s\n", prefix);
   return 0;
}
```

Q2: YACC Program to Validate if..then and do..while Statements
Lex Code:
lex
Copy code
```
%{
#include "y.tab.h"
%}
%%
"if"    { return IF; }
"then"  { return THEN; }
"do"    { return DO; }
"while" { return WHILE; }
[a-zA-Z]+ { return ID; }
[ \t]+; /* Ignore whitespace */
```

```
\n { return 0; }
. { return yytext[0]; }
%%
int yywrap() { return 1; }
YACC Code:
yacc
Copy code
%{
#include <stdio.h>
#include <stdlib.h>
int yylex();
int yyerror(const char *msg);
%}
%token IF THEN DO WHILE ID
%%
stmt : if_stmt
    | while_stmt
    ;
if_stmt : IF ID THEN { printf("Valid if-then statement\n"); }
      ;
while_stmt : DO ID WHILE { printf("Valid do-while statement\n"); }
         ;
%%
int yyerror(const char *msg) {
   fprintf(stderr, "Error: %s\n", msg);
   return 0;
}
int main() {
   printf("Enter a statement: ");
   yyparse();
   return 0;
}

#include <stdio.h>
#include <string.h>

#define SIZE 100

struct SymbolTableEntry {
   char name[50];
   char type[10];
   int scope;
} table[SIZE];

int count = 0;

void insert(char name[], char type[], int scope) {
   strcpy(table[count].name, name);
   strcpy(table[count].type, type);
   table[count].scope = scope;
   count++;
   printf("Inserted: %s\n", name);
}
```

```
int search(char name[]) {
  for (int i = 0; i < count; i++) {
    if (strcmp(table[i].name, name) == 0) {
      return i;
    }
  }
  return -1;
}

void display() {
  printf("\nSymbol Table:\n");
  printf("Name\tType\tScope\n");
  for (int i = 0; i < count; i++) {
    printf("%s\t%s\t%d\n", table[i].name, table[i].type, table[i].scope);
  }
}

int main() {
  int choice;
  char name[50], type[10];
  int scope, index;

  do {
    printf("\n1. Insert\n2. Search\n3. Display\n4. Exit\n");
    printf("Enter your choice: ");
    scanf("%d", &choice);

    switch (choice) {
      case 1:
        printf("Enter name, type, and scope: ");
        scanf("%s %s %d", name, type, &scope);
        insert(name, type, scope);
        break;
      case 2:
        printf("Enter name to search: ");
        scanf("%s", name);
        index = search(name);
        if (index != -1) {
          printf("Found %s at index %d\n", name, index);
        } else {
          printf("Not Found\n");
        }
        break;
      case 3:
        display();
        break;
      case 4:
        printf("Exiting...\n");
        break;
      default:
        printf("Invalid choice!\n");
    }
  } while (choice != 4);
```

```
    return 0;
}
```

```
#include <stdio.h>
#include <string.h>

void generate_intermediate_code(char expr[]) {
    char temp_var[3] = "t";
    int temp_count = 1;
    char result[3], op, arg1, arg2;

    printf("Intermediate Code:\n");
    for (int i = 0; expr[i] != '\0'; i++) {
        if (expr[i] == '+' || expr[i] == '-' || expr[i] == '*' || expr[i] == '/') {
            op = expr[i];
            arg1 = expr[i - 1];
            arg2 = expr[i + 1];
            sprintf(result, "%s%d", temp_var, temp_count++);
            printf("%s = %c %c %c\n", result, arg1, op, arg2);
            expr[i - 1] = result[1];
            expr[i] = expr[i + 1] = ' ';
        }
    }
}

int main() {
    char expr[50];
    printf("Enter an arithmetic expression (e.g., a+b*c): ");
    scanf("%s", expr);
    generate_intermediate_code(expr);
    return 0;
}
```

```
#include <stdio.h>
#include <string.h>
#include <ctype.h>

const char *keywords[] = {"int", "float", "if", "else", "return", "void", "while"};
const char *operators = "+-*/=<>!";
const char *delimiters = "(),;{}[]";

int is_keyword(const char *word) {
    for (int i = 0; i < sizeof(keywords) / sizeof(keywords[0]); i++) {
        if (strcmp(word, keywords[i]) == 0)
            return 1;
    }
    return 0;
```

```
}

int is_operator(char ch) {
  return strchr(operators, ch) != NULL;
}

int is_delimiter(char ch) {
  return strchr(delimiters, ch) != NULL;
}

void tokenize(char *line) {
  char *token = strtok(line, " \t\n");
  while (token != NULL) {
    if (is_keyword(token)) {
      printf("Keyword: %s\n", token);
    } else if (is_operator(token[0])) {
      printf("Operator: %s\n", token);
    } else if (is_delimiter(token[0])) {
      printf("Delimiter: %s\n", token);
    } else if (isdigit(token[0])) {
      printf("Number: %s\n", token);
    } else {
      printf("Identifier: %s\n", token);
    }
    token = strtok(NULL, " \t\n");
  }
}

int main() {
  FILE *file = fopen("input.c", "r");
  char line[256];

  if (!file) {
    perror("Error opening file");
    return 1;
  }

  printf("Tokens:\n");
  while (fgets(line, sizeof(line), file)) {
    tokenize(line);
  }

  fclose(file);
  return 0;
}
```

1. Write a LEX Program to identify and count the positive and negative floating points and zero.

Code:

```
%{
int pos_float_count = 0;
int neg_float_count = 0;
int zero_count = 0; %}
%%
[+-]?[0-9]*\.[0-9]+ {
```

```
if (yytext[0] == '-') {
neg_float_count++;
} else if (yytext[0] == '0' && (yytext[1] == '\0' || yytext[1] == '.'))
{ zero_count++;
} else { pos_float_count++; } }
[0] { zero_count++; } %%
int main() {
yylex();
printf("Positive Floating Points: %d\n", pos_float_count);
printf("Negative Floating Points: %d\n", neg_float_count);
printf("Zero Count: %d\n", zero_count); return 0;
}
int yywrap()
{ return 1; }
```

2. Write a LEX Program to identify whether the given symbol belongs to relational operator, arithmetic operator or others.

Code:

```
%{
#include <stdio.h> %}
%%
"+"|"-"|"*"|"/" {
printf("%s is an Arithmetic Operator\n", yytext);
}
"<"|">"|"<="|">="|"=="|"!=" {
printf("%s is a Relational Operator\n", yytext);
}
.{
printf("%s is an Other Symbol\n", yytext);
}
%%
int main() { yylex();
return 0; }
int yywrap() { return 1;
}
```

3. Write a LEX Program to identify whether the given email id is valid or not using file.

Code:

```
%{
#include <stdio.h> #include <regex.h>
int valid_email_count = 0; int invalid_email_count = 0;
void validate_email(const char *email) {
regex_t regex;
const char *pattern = "^[a-zA-Z0-9._%+-]+@[a-zA-Z0-9.-]+\\.[a-zA-Z]{2,}$"; int ret;
// Compile regular expression
ret = regcomp(&regex, pattern, REG_EXTENDED); if (ret) {
fprintf(stderr, "Could not compile regex\n");
return; }
// Execute regular expression
ret = regexec(&regex, email, 0, NULL, 0); if (!ret) {
```

```
printf("Valid Email: %s\n", email);
valid_email_count++; }else{
printf("Invalid Email: %s\n", email);
invalid_email_count++; }
// Free memory allocated to the regex
regfree(&regex); }
%}
%%
[a-zA-Z0-9._%+-]+@[a-zA-Z0-9.-]+\.[a-zA-Z]{2,}{ validate_email(yytext);
}
.*{
// For anything that doesn't match a valid email pattern printf("Invalid Email: %s\n", yytext); invalid_email_count++;
}
%%
int main() { yylex();
printf("\nTotal Valid Emails: %d\n", valid_email_count); printf("Total Invalid Emails: %d\n", invalid_email_count);
return 0;
}
int yywrap() { return 1;
}
```