**Problem (21)**

nano pos.l

%{

#include <stdio.h>

%}

%%

"the"|"The" { printf("Article: %s\n", yytext); }

"is"|"are"|"was"|"were" { printf("Verb: %s\n", yytext); }

"from"|"to"|"of"|"in"|"on" { printf("Preposition: %s\n", yytext); }

"it"|"he"|"she"|"they"|"we"|"I" { printf("Pronoun: %s\n", yytext); }

[.!?] { /\* Ignore punctuation \*/ }

[a-zA-Z]+ { printf("Noun/Verb: %s\n", yytext); }

[ \t\n]+ { /\* Ignore spaces \*/ }

%%

int main() {

yylex();

return 0;

}

flex pos.l # Generates lex.yy.c

gcc lex.yy.c -o pos -lfl # Compiles the LEX output

./pos # Runs the program

Dread it. Run from it. Destiny arrives all the same.

**Problem (22)**

nano pos.l

%{

#include <stdio.h>

%}

%%

"the"|"The" { printf("Article: %s\n", yytext); }

"is"|"are"|"was"|"were" { printf("Verb: %s\n", yytext); }

"from"|"to"|"of"|"in"|"on" { printf("Preposition: %s\n", yytext); }

"it"|"he"|"she"|"they"|"we"|"I"|"you"|"You" { printf("Pronoun: %s\n", yytext); }

"how"|"How"|"fine"|"thank"|"Thank" { printf("Adjective/Verb: %s\n", yytext); }

[.!?] { /\* Ignore punctuation \*/ }

[a-zA-Z]+ { printf("Noun/Verb: %s\n", yytext); }

[ \t\n]+ { /\* Ignore spaces \*/ }

%%

int main() {

yylex();

return 0;

}

flex pos.l

gcc lex.yy.c -o pos -lfl

./pos

Hello! How are you?

I’m fine, Thank You.

**Problem (23)**

nano pos.l

%{

#include <stdio.h>

%}

%%

"the"|"The" { printf("Article: %s\n", yytext); }

"is"|"are"|"was"|"were"|"has" { printf("Verb: %s\n", yytext); }

"from"|"to"|"of"|"in"|"on"|"for" { printf("Preposition: %s\n", yytext); }

"it"|"he"|"she"|"they"|"we"|"I"|"you"|"its" { printf("Pronoun: %s\n", yytext); }

"important"|"own"|"not" { printf("Adjective: %s\n", yytext); }

"thing"|"reason"|"questioning"|"curiosity"|"existing" { printf("Noun: %s\n", yytext); }

[.!?,] { /\* Ignore punctuation \*/ }

[a-zA-Z]+ { printf("Unknown: %s\n", yytext); }

[ \t\n]+ { /\* Ignore spaces \*/ }

%%

int main() {

yylex();

return 0;

}

flex pos.l

gcc lex.yy.c -o pos -lfl

./pos

The important thing is to not stop questioning,

Curiosity has its own reason for existing.

**Problem (24)**

nano pos.l

%{

#include <stdio.h>

%}

%%

"the"|"The" { printf("Article: %s\n", yytext); }

"is"|"are"|"was"|"were"|"give"|"kill"|"needs"|"can" { printf("Verb: %s\n", yytext); }

"from"|"to"|"of"|"in"|"on"|"for" { printf("Preposition: %s\n", yytext); }

"it"|"he"|"she"|"they"|"we"|"I"|"you" { printf("Pronoun: %s\n", yytext); }

"never"|"that"|"If" { printf("Conjunction/Qualifier: %s\n", yytext); }

"power"|"thing"|"cigarette"|"light" { printf("Noun: %s\n", yytext); }

[.!?,] { /\* Ignore punctuation \*/ }

[a-zA-Z]+ { printf("Unknown: %s\n", yytext); }

[ \t\n]+ { /\* Ignore whitespace \*/ }

%%

int main() {

yylex();

return 0;

}

flex pos.l

gcc lex.yy.c -o pos -lfl

./pos

If you never light the cigarette,

you never give the thing that can kill you the power it needs to kill you.

**Problem (25)**

nano pos.l

%{

#include <stdio.h>

%}

%%

"the"|"The" { printf("Article: %s\n", yytext); }

"to"|"than"|"before"|"after"|"be" { printf("Preposition: %s\n", yytext); }

"itself"|"it"|"one"|"Every" { printf("Pronoun: %s\n", yytext); }

"intelligent"|"wiser"|"more" { printf("Adjective: %s\n", yytext); }

"imagines"|"comes"|"went" { printf("Verb: %s\n", yytext); }

"generation" { printf("Noun: %s\n", yytext); }

"that"|"and" { printf("Conjunction: %s\n", yytext); }

[.!?,] { /\* Ignore punctuation \*/ }

[a-zA-Z]+ { printf("Unknown: %s\n", yytext); }

[ \t\n]+ { /\* Ignore whitespace \*/ }

%%

int main() {

yylex();

return 0;

}

flex pos.l

gcc lex.yy.c -o pos -lfl

./pos

Every generation imagines itself to be more intelligent than the one that went before it, and wiser than the one that comes after it.

**Problem (26)**

nano pos.l

%{

#include <stdio.h>

%}

%%

"have"|"divide" { printf("Verb: %s\n", yytext); }

"a"|"an"|"the" { printf("Article: %s\n", yytext); }

"toward"|"into"|"with"|"in"|"between" { printf("Preposition: %s\n", yytext); }

"Human"|"beings"|"instinct"|"thinking"|"urge"|"gap"|"groups"|"things" { printf("Noun: %s\n", yytext); }

"strong"|"dramatic"|"binary"|"basic"|"empty"|"distinct" { printf("Adjective: %s\n", yytext); }

"nothing"|"but" { printf("Qualifier: %s\n", yytext); }

[.!?,] { /\* Ignore punctuation \*/ }

[a-zA-Z]+ { printf("Unknown: %s\n", yytext); }

[ \t\n]+ { /\* Ignore whitespace \*/ }

%%

int main() {

yylex();

return 0;

}

flex pos.l

gcc lex.yy.c -o pos -lfl

./pos

Human beings have a strong dramatic instinct toward binary thinking, a basic urge to divide things into two distinct groups, with nothing but an empty gap in between.

**Problem (27)**

nano c\_lex.l

%{

#include <stdio.h>

%}

%%

"int" { printf("Keyword: %s\n", yytext); }

[0-9]+ { printf("Number: %s\n", yytext); }

[a-zA-Z\_][a-zA-Z0-9\_]\* { printf("Identifier: %s\n", yytext); }

"=" { printf("Assignment Operator: %s\n", yytext); }

"[-+\*/]" { printf("Operator: %s\n", yytext); }

[{},;] { printf("Symbol: %s\n", yytext); }

[ \t\n]+ { /\* Skip whitespace \*/ }

. { printf("Unknown: %s\n", yytext); }

%%

int main() {

yylex();

return 0;

}

flex c\_lex.l

gcc lex.yy.c -o c\_lex -lfl

./c\_lex

{ int m = 10, n = 2, o; o = m - n; }

**Problem (28)**

nano c\_lex.l

%{

#include <stdio.h>

%}

%%

"int"|"char" { printf("Keyword: %s\n", yytext); }

"printf" { printf("Function: %s\n", yytext); }

\".\*\" { printf("String: %s\n", yytext); }

[0-9]+ { printf("Number: %s\n", yytext); }

[a-zA-Z\_][a-zA-Z0-9\_]\* { printf("Identifier: %s\n", yytext); }

"=" { printf("Assignment Operator: %s\n", yytext); }

"[-+\*/]" { printf("Operator: %s\n", yytext); }

[{},;()] { printf("Symbol: %s\n", yytext); }

[ \t\n]+ { /\* Skip whitespace \*/ }

. { printf("Unknown: %s\n", yytext); }

%%

int main() {

yylex();

return 0;

}

flex c\_lex.l

gcc lex.yy.c -o c\_lex -lfl

./c\_lex

{

char = a;

printf("Hello World");

}

**Problem (29)**

nano c\_lex.l

%{

#include <stdio.h>

%}

%%

"int"|"char" { printf("Keyword: %s\n", yytext); }

"printf" { printf("Function: %s\n", yytext); }

\".\*\" { printf("String: %s\n", yytext); }

[0-9]+ { printf("Number: %s\n", yytext); }

[a-zA-Z\_][a-zA-Z0-9\_]\* { printf("Identifier: %s\n", yytext); }

"=" { printf("Assignment Operator: %s\n", yytext); }

"[-+\*/]" { printf("Operator: %s\n", yytext); }

[{},;()] { printf("Symbol: %s\n", yytext); }

[ \t\n]+ { /\* Ignore whitespace \*/ }

. { printf("Unknown: %s\n", yytext); }

%%

int main() {

yylex();

return 0;

}

flex c\_lex.l

gcc lex.yy.c -o c\_lex -lfl

./c\_lex

{

int d = 10, t = 2;

int s;

s = d / t;

}

**Problem (30)**

nano c\_lex.l

%{

#include <stdio.h>

%}

%%

"int"|"char"|"float" { printf("Keyword: %s\n", yytext); }

"printf" { printf("Function: %s\n", yytext); }

\".\*\" { printf("String: %s\n", yytext); }

[0-9]+ { printf("Number: %s\n", yytext); }

[a-zA-Z\_][a-zA-Z0-9\_]\* { printf("Identifier: %s\n", yytext); }

"=" { printf("Assignment Operator: %s\n", yytext); }

"[-+\*/]" { printf("Operator: %s\n", yytext); }

[{},;()] { printf("Symbol: %s\n", yytext); }

[ \t\n]+ { /\* Ignore whitespace \*/ }

. { printf("Unknown: %s\n", yytext); }

%%

int main() {

yylex();

return 0;

}

flex c\_lex.l

gcc lex.yy.c -o c\_lex -lfl

./c\_lex

{

int a = 3;

int b = 4;

float c;

c = (a\*a + b\*b) \* 2; }

**Problem (31)**

nano c\_lex.l

%{

#include <stdio.h>

%}

%%

"int"|"char"|"float" { printf("Keyword: %s\n", yytext); }

"printf" { printf("Function: %s\n", yytext); }

\".\*\" { printf("String: %s\n", yytext); }

[0-9]+ { printf("Number: %s\n", yytext); }

[a-zA-Z\_][a-zA-Z0-9\_]\* { printf("Identifier: %s\n", yytext); }

"=" { printf("Assignment Operator: %s\n", yytext); }

"[-+\*/]" { printf("Operator: %s\n", yytext); }

[{},;()] { printf("Symbol: %s\n", yytext); }

[ \t\n]+ { /\* Ignore whitespace \*/ }

. { printf("Unknown: %s\n", yytext); }

%%

int main() {

yylex();

return 0;

}

flex c\_lex.l

gcc lex.yy.c -o c\_lex -lfl

./c\_lex

{

int total = 100;

int i = 10;

printf("The value of total and i is : %d, %d", total, i);

}

**Problem (32) - Problem (35)**

**nano expr.l**

%{

#include "y.tab.h"

%}

%%

[0-9]+\.[0-9]+ { yylval.dval = atof(yytext); return NUM; }

[0-9]+ { yylval.dval = atoi(yytext); return NUM; }

"(" { return '('; }

")" { return ')'; }

"+" { return '+'; }

"-" { return '-'; }

"\*" { return '\*'; }

"/" { return '/'; }

[ \t\n]+ { /\* skip whitespace \*/ }

. { return yytext[0]; }

%%

int yywrap() {

return 1;

}

**nano expr.y**

%{

#include <stdio.h>

#include <stdlib.h>

int yylex(void);

int yyerror(const char \*s);

%}

%union {

double dval;

}

%token <dval> NUM

%type <dval> expr

%left '+' '-'

%left '\*' '/'

%%

expr: expr '+' expr { $$ = $1 + $3; printf("= %f\n", $$); }

| expr '-' expr { $$ = $1 - $3; printf("= %f\n", $$); }

| expr '\*' expr { $$ = $1 \* $3; printf("= %f\n", $$); }

| expr '/' expr {

if ($3 == 0) { printf("Divide by zero!\n"); exit(1); }

$$ = $1 / $3; printf("= %f\n", $$);

}

| '(' expr ')' { $$ = $2; }

| NUM { $$ = $1; }

;

%%

int main() {

printf("Enter expression:\n");

yyparse();

return 0;

}

int yyerror(const char \*s) {

fprintf(stderr, "Error: %s\n", s);

return 0;

}

yacc -d expr.y

flex expr.l

gcc y.tab.c lex.yy.c -o expr -lm

./expr

0.33 \* 12 - 4 - 4 + (3 \* 2)

**Problem (36)**

**nano varname.l**

%{

#include "y.tab.h"

%}

%%

[a-zA-Z\_][a-zA-Z0-9\_]\* { yylval.str = strdup(yytext); return IDENTIFIER; }

\n { return 0; } // end of input

. { return yytext[0]; }

%%

int yywrap() {

return 1;

}

**nano varname.y**

%{

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

int yylex(void);

int yyerror(const char \*s);

%}

%union {

char \*str;

}

%token <str> IDENTIFIER

%%

input: IDENTIFIER { printf("Valid identifier: %s\n", $1); }

;

%%

int main() {

printf("Enter a variable name: ");

yyparse();

return 0;

}

int yyerror(const char \*s) {

printf("Invalid identifier\n");

return 0;

}

yacc -d varname.y

flex varname.l

gcc y.tab.c lex.yy.c -o varname -lfl

./varname

**Problem (37)**

nano caseconv.l

%{

#include "y.tab.h"

#include <string.h>

%}

%%

[a-zA-Z]+ { yylval.str = strdup(yytext); return WORD; }

\n { return 0; }

. { return yytext[0]; }

%%

int yywrap() {

return 1;

}

nano caseconv.y

%{

#include <stdio.h>

#include <stdlib.h>

#include <ctype.h>

#include <string.h>

int yylex(void);

int yyerror(const char \*s);

%}

%union {

char \*str;

}

%token <str> WORD

%%

input: WORD {

printf("Original: %s\n", $1);

printf("Uppercase: ");

for (int i = 0; $1[i]; i++) {

printf("%c", toupper($1[i]));

}

printf("\nLowercase: ");

for (int i = 0; $1[i]; i++) {

printf("%c", tolower($1[i]));

}

printf("\n");

}

;

%%

int main() {

printf("Enter a word: ");

yyparse();

return 0;

}

int yyerror(const char \*s) {

printf("Error: %s\n", s);

return 0;

}

yacc -d caseconv.y

flex caseconv.l

gcc y.tab.c lex.yy.c -o caseconv -lfl

./caseconv

**Problem (38)**

nano casecheck.l

%{

#include "y.tab.h"

#include <string.h>

%}

%%

[a-zA-Z]+ { yylval.str = strdup(yytext); return WORD; }

\n { return 0; }

. { return yytext[0]; }

%%

int yywrap() {

return 1;

}

nano casecheck.y

%{

#include <stdio.h>

#include <ctype.h>

#include <string.h>

#include <stdlib.h>

int yylex(void);

int yyerror(const char \*s);

%}

%union {

char \*str;

}

%token <str> WORD

%%

input: WORD {

int upper = 0, lower = 0;

for (int i = 0; $1[i]; i++) {

if (isupper($1[i])) upper++;

else if (islower($1[i])) lower++;

}

if (upper > 0 && lower == 0)

printf("UPPERCASE\n");

else if (lower > 0 && upper == 0)

printf("lowercase\n");

else

printf("Mixed Case\n");

}

;

%%

int main() {

printf("Enter a word: ");

yyparse();

return 0;

}

int yyerror(const char \*s) {

printf("Error: %s\n", s);

return 0;

}

yacc -d casecheck.y

flex casecheck.l

gcc y.tab.c lex.yy.c -o casecheck -lfl

./casecheck

**Problem (39)**

nano digitcheck.l

%{

#include "y.tab.h"

%}

%%

[0-9] { yylval.num = yytext[0]; return DIGIT; }

[a-zA-Z]+ { return WORD; }

\n { return 0; }

. { return OTHER; }

%%

int yywrap() {

return 1;

}

nano digitcheck.y

%{

#include <stdio.h>

int yylex(void);

int yyerror(const char \*s);

%}

%union {

char num;

}

%token <num> DIGIT

%token WORD OTHER

%%

input: DIGIT { printf("Valid Digit: %c\n", $1); }

| WORD { printf("Not a digit: It's a word\n"); }

| OTHER { printf("Not a digit: Invalid input\n"); }

;

%%

int main() {

printf("Enter a single character: ");

yyparse();

return 0;

}

int yyerror(const char \*s) {

printf("Error: %s\n", s);

return 0;

}

yacc -d digitcheck.y

flex digitcheck.l

gcc y.tab.c lex.yy.c -o digitcheck -lfl

./digitcheck

**Problem (40)**

nano mathfunc.l

%{

#include "y.tab.h"

#include <stdlib.h>

%}

%%

"sin" { return SIN; }

"cos" { return COS; }

"log" { return LOG; }

[0-9]+ { yylval.num = atoi(yytext); return NUMBER; }

[ \t\n]+ { /\* Skip whitespace \*/ }

. { return yytext[0]; }

%%

int yywrap() {

return 1;

}

nano mathfunc.y

%{

#include <stdio.h>

#include <math.h>

int yylex(void);

int yyerror(const char \*s);

%}

%union {

int num;

}

%token <num> NUMBER

%token SIN COS LOG

%%

input: SIN NUMBER {

printf("sin(%d) = %.4f\n", $2, sin($2 \* M\_PI / 180)); // convert to radians

}

| COS NUMBER {

printf("cos(%d) = %.4f\n", $2, cos($2 \* M\_PI / 180));

}

| LOG NUMBER {

if ($2 > 0)

printf("log(%d) = %.4f\n", $2, log($2));

else

printf("Logarithm undefined for non-positive values.\n");

}

;

%%

int main() {

printf("Enter input (e.g., sin 30): ");

yyparse();

return 0;

}

int yyerror(const char \*s) {

printf("Error: %s\n", s);

return 0;

}

yacc -d mathfunc.y

flex mathfunc.l

gcc y.tab.c lex.yy.c -o mathfunc -lm

./mathfunc

**Problem (41) - (48)**

**package** lpcc;

**import** java.util.\*;

**public** **class** TACGenerator {

**static** **int** *tempCount* = 1;

**static** List<String> *tac* = **new** ArrayList<>();

**static** String newTemp() {

**return** "t" + (*tempCount*++);

}

**static** String generate(String expr) {

Stack<String> operands = **new** Stack<>();

Stack<Character> operators = **new** Stack<>();

**for** (**int** i = 0; i < expr.length(); i++) {

**char** ch = expr.charAt(i);

**if** (ch == ' ') **continue**;

**if** (Character.*isLetterOrDigit*(ch)) {

operands.push(String.*valueOf*(ch));

} **else** **if** (ch == '+' || ch == '-' || ch == '\*' || ch == '/' || ch == '^') {

**while** (!operators.isEmpty() && *precedence*(operators.peek()) >= *precedence*(ch)) {

*makeTAC*(operators.pop(), operands);

}

operators.push(ch);

} **else** **if** (ch == '(') {

operators.push(ch);

} **else** **if** (ch == ')') {

**while** (!operators.isEmpty() && operators.peek() != '(') {

*makeTAC*(operators.pop(), operands);

}

operators.pop(); // remove '('

}

}

**while** (!operators.isEmpty()) {

*makeTAC*(operators.pop(), operands);

}

**return** operands.pop();

}

**static** **void** makeTAC(**char** op, Stack<String> operands) {

String b = operands.pop();

String a = operands.pop();

String t = *newTemp*();

*tac*.add(t + " = " + a + " " + op + " " + b);

operands.push(t);

}

**static** **int** precedence(**char** op) {

**switch** (op) {

**case** '^': **return** 3;

**case** '\*': **case** '/': **return** 2;

**case** '+': **case** '-': **return** 1;

}

**return** 0;

}

**public** **static** **void** main(String[] args) {

Scanner sc = **new** Scanner(System.***in***);

System.***out***.print("Enter LHS variable: ");

String lhs = sc.nextLine();

System.***out***.print("Enter arithmetic expression: ");

String expr = sc.nextLine().replace(" ", ""); // remove spaces

String resultTemp = *generate*(expr);

*tac*.add(lhs + " = " + resultTemp);

System.***out***.println("\nThree Address Code:");

**for** (String line : *tac*) {

System.***out***.println(line);

}

sc.close();

}

}