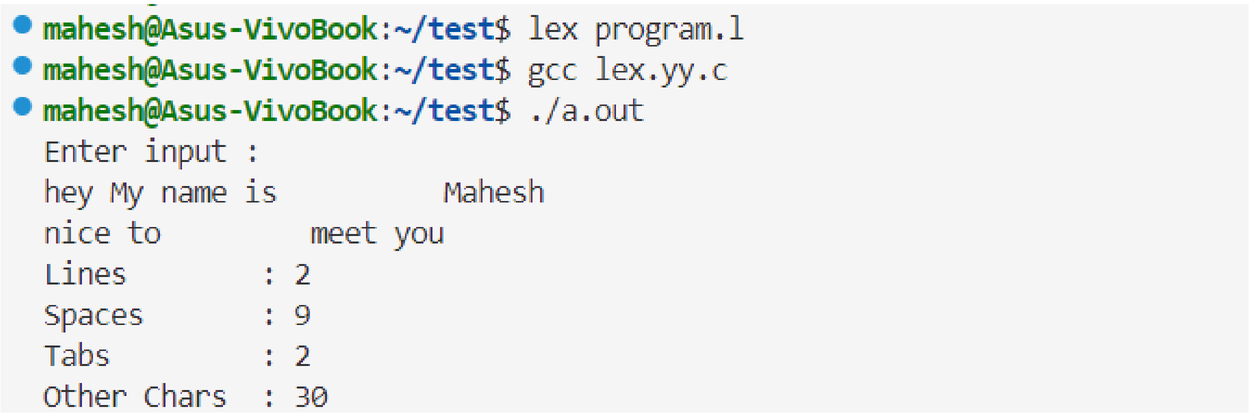
Name : Mahesh Semwal
Section : A-RQ
University roll no. : 2022037

PROBLEM STATEMENT 2:

Design a LEX Code to count the number of lines, space, tab-meta character, and rest of characters in each Input pattern.

```
%{
#include <stdio.h>
int lineCount = 0;
int spaceCount = 0;
int tabCount = 0;
int otherCount = 0;
%}

%%
\n          { lineCount++; }
[ ]         { spaceCount++; }
[\t]        { tabCount++; }
.           { otherCount++; }
%%

int main(int argc, char **argv) {
   printf("Enter input :\n");
   yylex();
   printf("Lines       : %d\n", lineCount);
   printf("Spaces      : %d\n", spaceCount);
   printf("Tabs        : %d\n", tabCount);
   printf("Other Chars  : %d\n", otherCount);
   return 0;
}


int yywrap() {
   return 1;
}
```

OUTPUT



```
maheshh@Asus-VivoBook:~/test$ lex program.l
maheshh@Asus-VivoBook:~/test$ gcc lex.yy.c
maheshh@Asus-VivoBook:~/test$ ./a.out
Enter input :
hey My name is          Mahesh
nice to         meet you
Lines        : 2
Spaces       : 9
Tabs         : 2
Other Chars  : 30
```

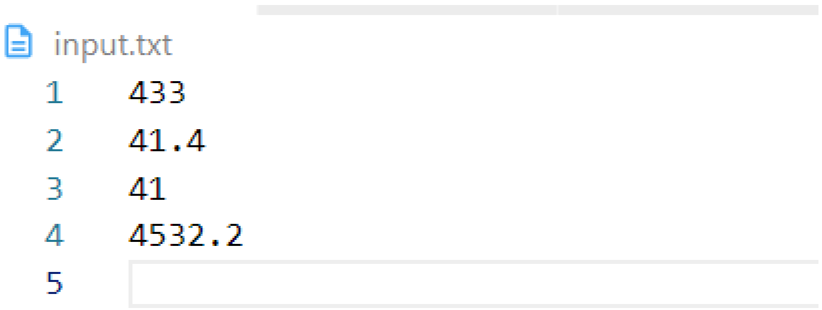
Name : Mahesh Semwal
Section : A-RQ
University roll no. : 2022037

PROBLEM STATEMENT 4:

Design a LEX Code to identify and print integer and float value in a given input pattern.

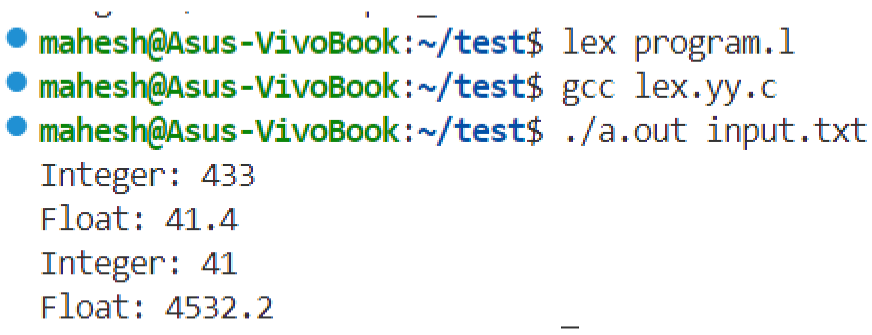
```
%{
#include <stdio.h>
FILE *yyin;
%}
%%
[0-9]*\.[0-9]+    { printf("Float: %s\n", yytext); }
[0-9]+\.[0-9]*    { printf("Float: %s\n", yytext); }
[0-9]+            { printf("Integer: %s\n", yytext); }
[ \t\n]+          { /* Skip whitespace */ }
.                 { /* Ignore other characters */ }
%%
int yywrap() {
  return 1;
}
int main(int argc, char *argv[]) {
  if (argc < 2) {
    fprintf(stderr, "Usage: %s <input_file>\n", argv[0]);
    return 1;}
  yyin = fopen(argv[1], "r");
  if (!yyin) {
    perror("Error opening input file");
    return 1;
  }
  yylex();
  fclose(yyin);
  return 0;}
```

INPUT FILE:



```
input.txt
433
41.4
41
4532.2
```

OUTPUT:



```
mahesh@Asus-VivoBook:~/test$ lex program.l
mahesh@Asus-VivoBook:~/test$ gcc lex.yy.c
mahesh@Asus-VivoBook:~/test$ ./a.out input.txt
Integer: 433
Float: 41.4
Integer: 41
Float: 4532.2
```

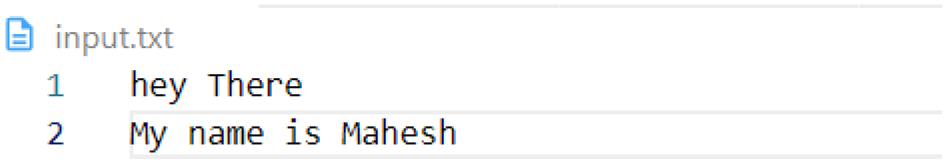
Name : Mahesh Semwal
Section : A-RQ
University roll no. : 2022037

PROBLEM STATEMENT 6:

Design a LEX Code to count and print the total number of characters, words, white spaces and lines in a given file named as ‘Input.txt’.

```
%{

#include <stdio.h>

int charCount = 0;

int wordCount = 0;

int whiteSpaceCount = 0;

int lineCount = 0;

%}


%%


[ \t]+         { charCount += yyleng; whiteSpaceCount += yyleng; }

\n             { charCount += 1; lineCount++; }

[a-zA-Z0-9_]+  { charCount += yyleng; wordCount++; }

.              { charCount += 1; }


%%


int main() {

    FILE *file = fopen("input.txt", "r");

    if (!file) {

        perror("Error opening Input.txt");

        return 1;

    }
    yyin = file;
    yylex();
    fclose(file);

    printf("Total characters: %d\n", charCount);
    printf("Total words: %d\n", wordCount);
    printf("Total white spaces: %d\n", whiteSpaceCount);
    printf("Total lines: %d\n", lineCount);

    return 0;
}

int yywrap() {
    return 1;
}
```

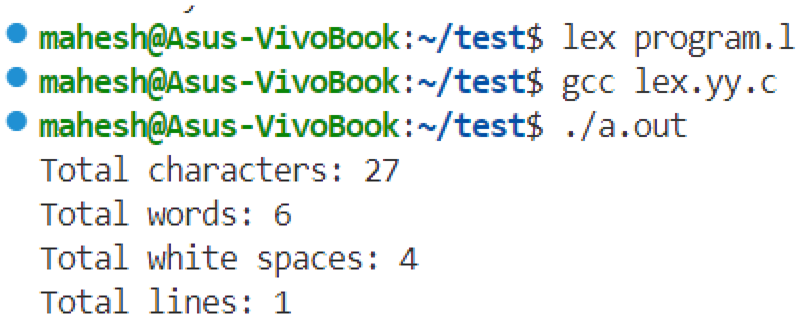
INPUT FILE



input.txt

```
hey There
My name is Mahesh
```

OUTPUT



```
mahesh@Asus-VivoBook:~/test$ lex program.l
mahesh@Asus-VivoBook:~/test$ gcc lex.yy.c
mahesh@Asus-VivoBook:~/test$ ./a.out
Total characters: 27
Total words: 6
Total white spaces: 4
Total lines: 1
```

Name : Mahesh Semwal
Section : A-RQ
University roll no. : 2022037

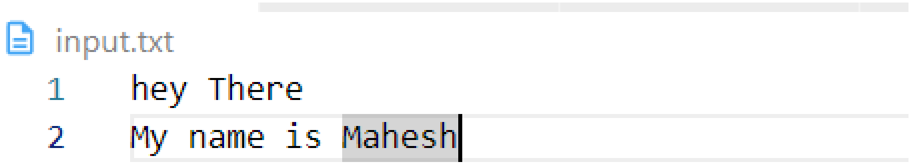
PROBLEM STATEMENT 7:

Design a LEX Code to replace all the white spaces of ‘Input.txt’ file by a single blank character and store the output in ‘Output.txt’ file.

```
%{

#include <stdio.h>

FILE *outFile;

%}


%%

[ \t]+    { fputc('@', outFile); }

\n   { fputc('\n', outFile); }

.              { fputc(yytext[0], outFile); }

%%


int main() {

   FILE *inFile = fopen("input.txt", "r");

   if (!inFile) {

      perror("Error opening Input.txt");

      return 1;

   }

   outFile = fopen("output.txt", "w");

   if (!outFile) {

      perror("Error opening Output.txt");

      fclose(inFile);

      return 1;

   }
```

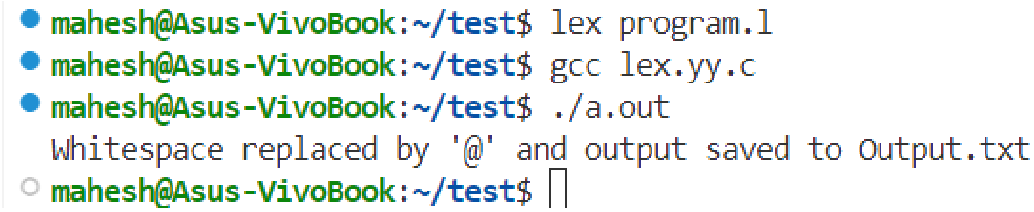
```
    yyin = inFile;
    yylex();
    fclose(inFile);
    fclose(outFile);
    printf("Whitespace replaced by '@' and output saved to Output.txt\n");
    return 0;
}

int yywrap() {
    return 1;
}
```

INPUT FILE



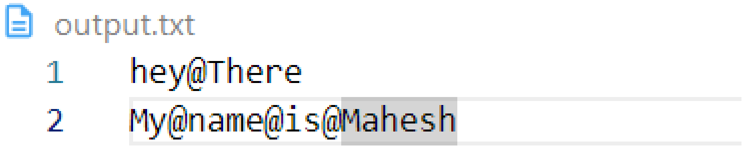
```
hey There
My name is Mahesh
```

TERMINAL



```
mahesh@Asus-VivoBook:~/test$ lex program.l
mahesh@Asus-VivoBook:~/test$ gcc lex.yy.c
mahesh@Asus-VivoBook:~/test$ ./a.out
Whitespace replaced by '@' and output saved to Output.txt
mahesh@Asus-VivoBook:~/test$ 
```

OUTPUT FILE



```
hey@There
My@name@is@Mahesh
```

Name : Mahesh Semwal
Section : A-RQ
University roll no. : 2022037

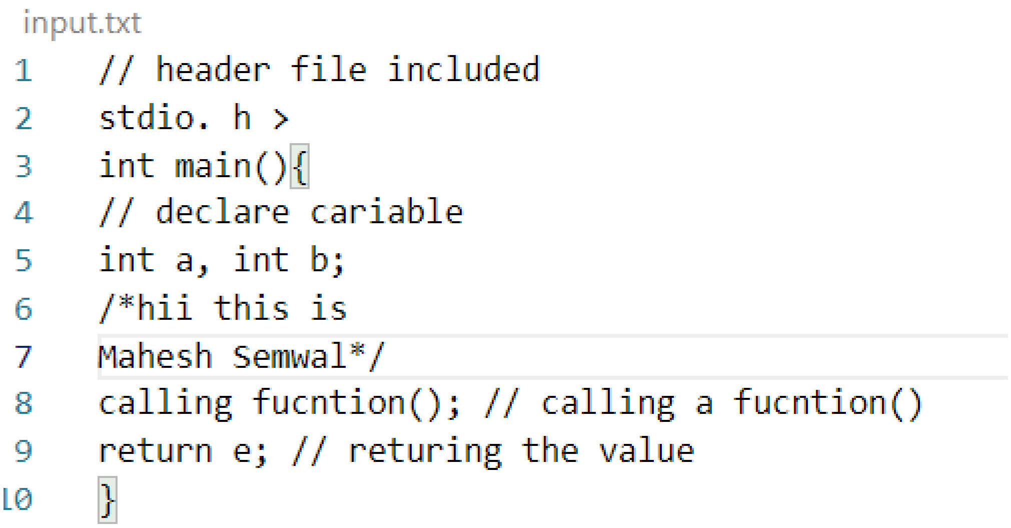
PROBLEM STATEMENT 8:

Design a LEX Code to remove the comments from any C-Program (in.c) given at run-time and store into ‘out.c’ file.

```
%{

#include <stdio.h>

FILE *yyin;

FILE *yyout;

%}

%%

\/\/[^\n]* ;

\/\*[^*]*\*\/ ;

.   { fputc(yytext[0], yyout); }

%%

int main() {

   yyin = fopen("input.txt", "r");

   if (!yyin) {

       perror("Failed to open input file");

       return 1;

   }


   yyout = fopen("output.txt", "w");

   if (!yyout) {

       perror("Failed to open output file");

       fclose(yyin);

       return 1;

   }
```

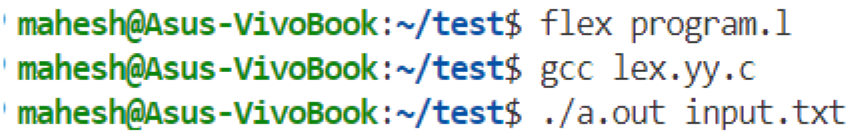
```
    yylex();
    fclose(yyin);
    fclose(yyout);
    return 0;
}
int yywrap() {
    return 1;
}
```

INPUT FILE



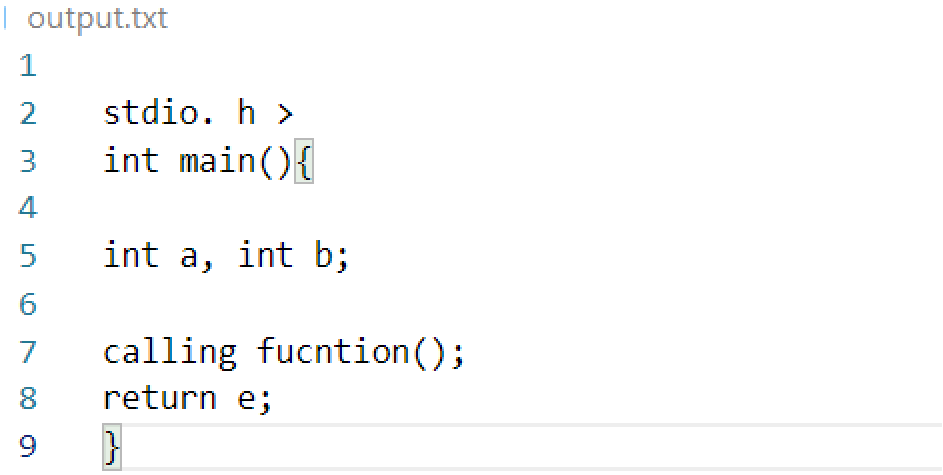
input.txt

```
// header file included
stdio. h >
int main(){
// declare cariable
int a, int b;
/*hii this is
Mahesh Semwal*/
calling fucntion(); // calling a fucntion()
return e; // returing the value
}
```



```
mahesh@Asus-VivoBook:~/test$ flex program.l
mahesh@Asus-VivoBook:~/test$ gcc lex.yy.c
mahesh@Asus-VivoBook:~/test$ ./a.out input.txt
```

OUTPUT FILE:



```

stdio. h >
int main(){

int a, int b;

calling fucntion();
return e;
}
```

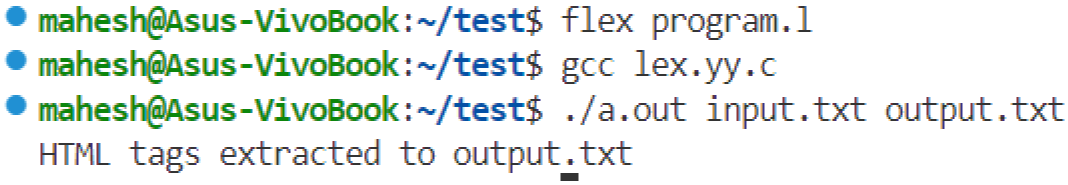
Name : Mahesh Semwal
Section : A-RQ
University roll no. : 2022037

PROBLEM STATEMENT 9:

Design a LEX Code to extract all html tags in the HTML file given at run time and store into text file given at run time.

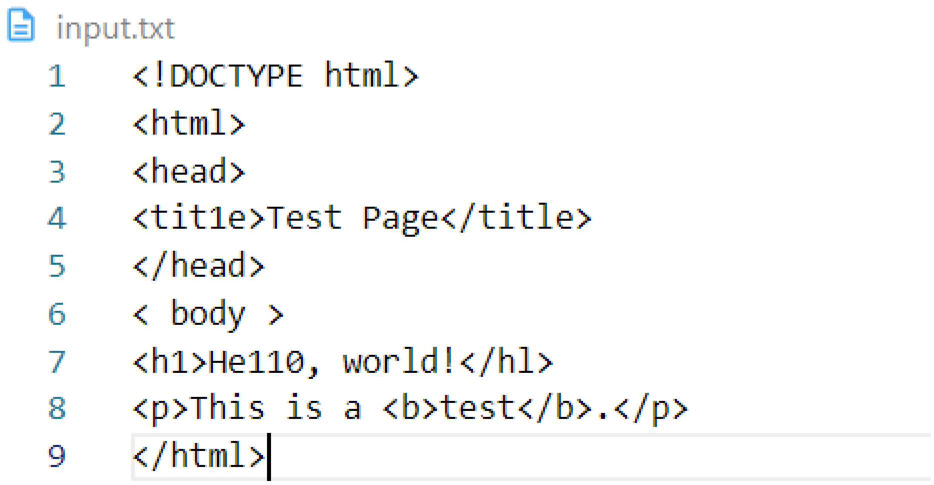
```
%{

#include <stdio.h>

#include <stdlib.h>

FILE *outFile;

%}

%%

\<[^>]+\>            { fprintf(outFile, "%s\n", yytext); }

.|\n                ;

%%

int main(int argc, char *argv[]) {

    if (argc != 3) {

        printf("Usage: %s input.html output.txt\n", argv[0]);

        exit(1);

    }

    FILE *inFile = fopen(argv[1], "r");

    if (!inFile) {

        perror("Error opening input file");

        exit(1);

    }

    outFile = fopen(argv[2], "w");

    if (!outFile) {

        perror("Error opening output file");
        fclose(inFile);
        exit(1);
    }
    yyin = inFile;
    yylex();

    fclose(inFile);
    fclose(outFile);
    printf("HTML tags extracted to %s\n", argv[2]);
    return 0;
}

int yywrap() {
    return 1;
}
```

TERMINAL



```
● mahesh@Asus-VivoBook:~/test$ flex program.l
● mahesh@Asus-VivoBook:~/test$ gcc lex.yy.c
● mahesh@Asus-VivoBook:~/test$ ./a.out input.txt output.txt
  HTML tags extracted to output.txt
```

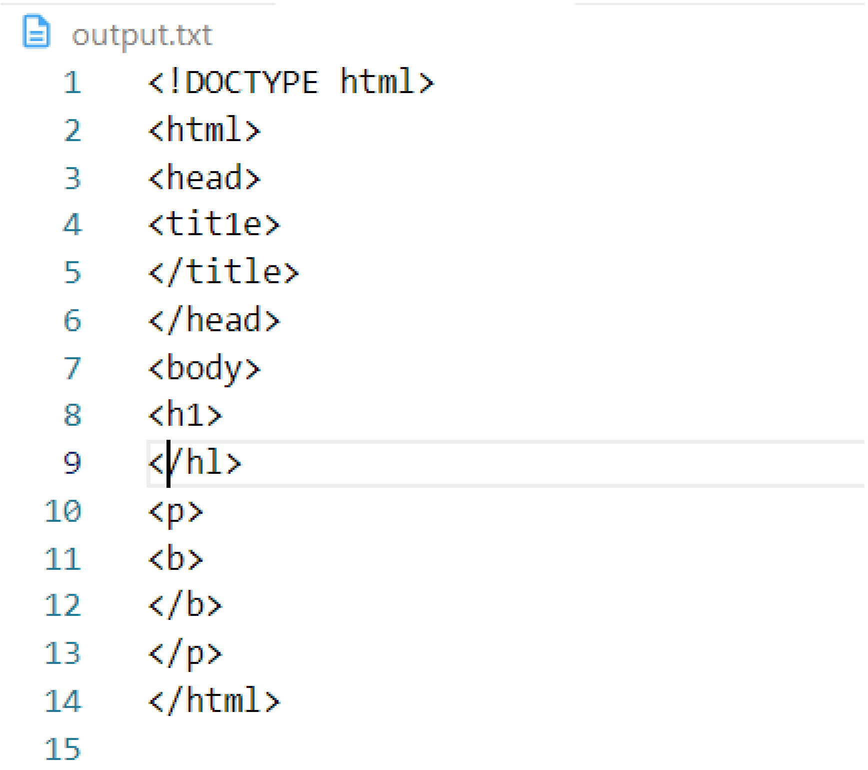
INPUT FILE



input.txt

```
<!DOCTYPE html>
<html>
<head>
<tit1e>Test Page</title>
</head>
< body >
<h1>He110, world!</hl>
<p>This is a <b>test</b>.</p>
</html>
```

OUTPUT FILE



output.txt

```
<!DOCTYPE html>
<html>
<head>
<tit1e>
</title>
</head>
<body>
<h1>
</hl>
<p>
<b>
</b>
</p>
</html>

```

Name : Mahesh Semwal
Section : A-RQ
University roll no. : 2022037

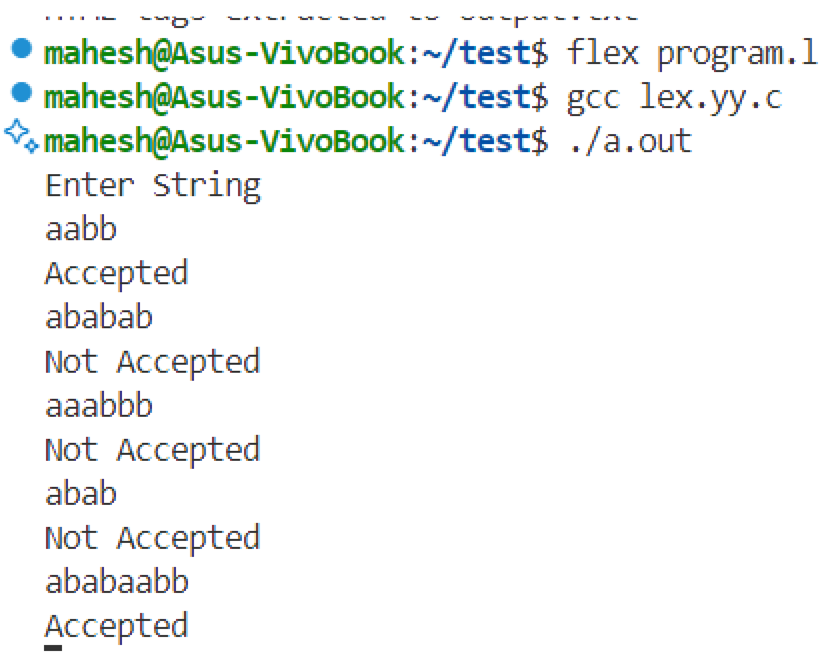
PROBLEM STATEMENT 11:

Design a DFA in LEX Code which accepts string containing third last element ‘a’ over the input alphabet {a, b}.

```
%{
%}
%s A B C D E F G DEAD
%%
<INITIAL>b BEGIN INITIAL;
<INITIAL>a BEGIN A;
<INITIAL>[^ab\n] BEGIN DEAD;
<INITIAL>\n BEGIN INITIAL; {printf("Not Accepted\n");}
<A>b BEGIN F;
<A>a BEGIN B;
<A>[^ab\n] BEGIN DEAD;
<A>\n BEGIN INITIAL; {printf("Not Accepted\n");}
<B>b BEGIN D;
<B>a BEGIN C;
<B>[^ab\n] BEGIN DEAD;
<B>\n BEGIN INITIAL; {printf("Not Accepted\n");}
<C>b BEGIN D;
<C>a BEGIN C;
<C>[^ab\n] BEGIN DEAD;
<C>\n BEGIN INITIAL; {printf("Accepted\n");}
<D>b BEGIN G;
<D>a BEGIN E;
<D>[^ab\n] BEGIN DEAD;
```

```
<D>\n BEGIN INITIAL; {printf("Accepted\n");}
<E>b BEGIN F;
<E>a BEGIN B;
<E>[^ab\n] BEGIN DEAD;
<E>\n BEGIN INITIAL; {printf("Accepted\n");}
<F>b BEGIN G;
<F>a BEGIN E;
<F>[^ab\n] BEGIN DEAD;
<F>\n BEGIN INITIAL; {printf("Not Accepted\n");}
<G>b BEGIN INITIAL;
<G>a BEGIN A;
<G>[^ab\n] BEGIN DEAD;
<G>\n BEGIN INITIAL; {printf("Accepted\n");}
<DEAD>[^\n] BEGIN DEAD;
<DEAD>\n BEGIN INITIAL; {printf("Invalid\n");}
%%
int yywrap(){
 return 1;
}
int main(){
 printf("Enter String\n");
 yylex();
 return 0;
}
```

OUTPUT



```
HTML tags extracted to output.txt
● mahesh@Asus-VivoBook:~/test$ flex program.l
● mahesh@Asus-VivoBook:~/test$ gcc lex.yy.c
  mahesh@Asus-VivoBook:~/test$ ./a.out
  Enter String
  aabb
  Accepted
  ababab
  Not Accepted
  aaabbb
  Not Accepted
  abab
  Not Accepted
  ababaabb
  Accepted
```

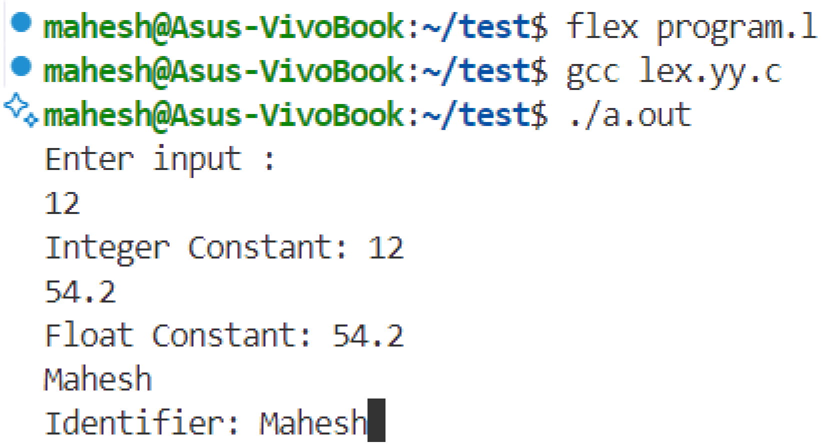
Name : Mahesh Semwal
Section : A-RQ
University roll no. : 2022037

PROBLEM STATEMENT 12:

Design a DFA in LEX Code to identify and print integer & float constants and identifier.

```
%{
#include <stdio.h>
%}
%s A
%%
<INITIAL>[0-9]+.[0-9]+  {printf("Float Constant: %s", yytext);}
<INITIAL>[0-9]+ { printf("Integer Constant: %s", yytext);}
<INITIAL>[a-zA-Z_][a-zA-Z0-9_]* { printf("Identifier: %s", yytext);}
<INITIAL>\n  {}
<INITIAL>. BEGIN A;
<A>\n  {printf("Invalid");}
%%


int main() {
   printf("Enter input :\n");
   yylex();
   return 0;
}
int yywrap() {
   return 1;
}
```

OUTPUT



```
mahesh@Asus-VivoBook:~/test$ flex program.l
mahesh@Asus-VivoBook:~/test$ gcc lex.yy.c
mahesh@Asus-VivoBook:~/test$ ./a.out
Enter input :
12
Integer Constant: 12
54.2
Float Constant: 54.2
Mahesh
Identifier: Mahesh
```

Name : Mahesh Semwal
Section : A-RQ
University roll no. : 2022037

PROBLEM STATEMENT 13:

Design YACC/LEX code to recognize the valid string from the language L= {anbn |n>=1}.

LEX FILE

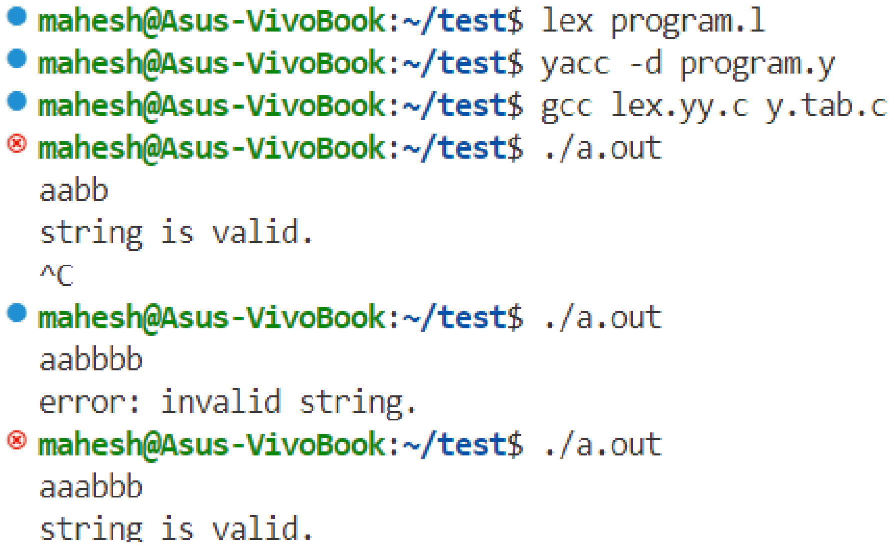
```
%{
#include<stdio.h>
#include "y.tab.h"
%}


%%
a {return A;}
b {return B;}
\n {return '\n';}
. ;
%%
int yywrap(){
   return 1;
}
```

YACC FILE

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

```
%token A B
%start E
%%
E : S '\n'  {printf("string is valid.\n");}
;
S:A B
|
A S B
;
%%
int main(){
   yyparse();
   return 0;
}
void yyerror(const char *s){
   fprintf(stderr,"error: invalid string.\n");
}
```

OUTPUT



```
● mahesh@Asus-VivoBook:~/test$ lex program.l
● mahesh@Asus-VivoBook:~/test$ yacc -d program.y
● mahesh@Asus-VivoBook:~/test$ gcc lex.yy.c y.tab.c
⊗ mahesh@Asus-VivoBook:~/test$ ./a.out
  aabb
  string is valid.
  ^C
● mahesh@Asus-VivoBook:~/test$ ./a.out
  aabbbb
  error: invalid string.
⊗ mahesh@Asus-VivoBook:~/test$ ./a.out
  aaabbb
  string is valid.
```

Name : Mahesh Semwal
Section : A-RQ
University roll no. : 2022037

PROBLEM STATEMENT 14:

Design YACC/LEX code to recognize valid arithmetic expression with operators +, -, * and /.

LEX FILE

```
%{
#include "y.tab.h"
#include <stdlib.h>
extern int yylval;
int yywrap();
%}
%%
[0-9]       { yylval = atoi(yytext); return ID; }
[\n]        { return '\n'; }
[+\-*/()]         { return yytext[0]; }


%%
int yywrap() { return 1; }
```

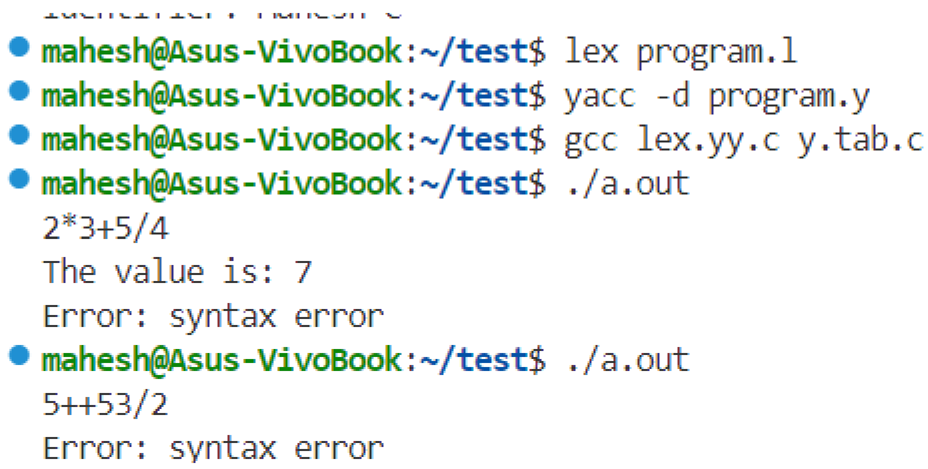
YACC FILE

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


int yylex(void);
void yyerror(const char *s);
%}
```

```
%token ID
%left '+' '-'
%left '*' '/'
%right '^'

%start S

%%

S : E '\n'  {printf("The value is: %d",$1); };

E : E '+' E   { $$=$1+$3; }
  | E '-' E   { $$=$1-$3; }
  | E '*' E   { $$=$1*$3; }
  | E '/' E   { $$=$1/$3; }
  | '(' E ')' { $$=$2; }
  | ID        { $$=$1; }
  ;

%%

int main() {
    yyparse();
    return 0;
}
```

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

OUTPUT



```
  identifier: mahesh c
● mahesh@Asus-VivoBook:~/test$ lex program.l
● mahesh@Asus-VivoBook:~/test$ yacc -d program.y
● mahesh@Asus-VivoBook:~/test$ gcc lex.yy.c y.tab.c
● mahesh@Asus-VivoBook:~/test$ ./a.out
  2*3+5/4
  The value is: 7
  Error: syntax error
● mahesh@Asus-VivoBook:~/test$ ./a.out
  5++53/2
  Error: syntax error
```

Name : Mahesh Semwal
Section : A-RQ
University roll no. : 2022037

PROBLEM STATEMENT 15:

Design YACC/LEX code to evaluate the arithmetic expression involving operators +, -, * and / with operator precedence grammar.

LEX FILE

```
%{
#include "y.tab.h"
#include <stdlib.h>
extern int yylval;
int yywrap();
%}
%%
[0-9]        { yylval = atoi(yytext); return ID; }
[\n]        { return '\n'; }
[+\-*/()]          { return yytext[0]; }


%%
int yywrap() { return 1; }
```

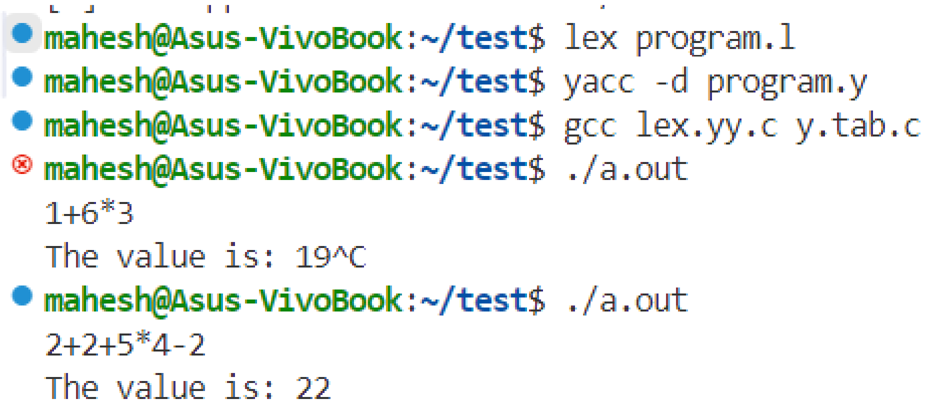
YACC FILE

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


int yylex(void);
void yyerror(const char *s);
%}
```

```
%token ID
%left '+' '-'
%left '*' '/'
%right '^'

%start S

%%

S : E '\n'  {printf("The value is: %d",$1); };

E : E '+' E   { $$=$1+$3; }
  | E '-' E   { $$=$1-$3; }
  | E '*' E   { $$=$1*$3; }
  | E '/' E   { $$=$1/$3; }
  | '(' E ')' { $$=$2; }
  | ID        { $$=$1; }
  ;

%%

int main() {
    yyparse();
    return 0;
}
```

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

OUTPUT



```
mahesh@Asus-VivoBook:~/test$ lex program.l
mahesh@Asus-VivoBook:~/test$ yacc -d program.y
mahesh@Asus-VivoBook:~/test$ gcc lex.yy.c y.tab.c
mahesh@Asus-VivoBook:~/test$ ./a.out
1+6*3
The value is: 19^C
mahesh@Asus-VivoBook:~/test$ ./a.out
2+2+5*4-2
The value is: 22
```

VALUE ADDITION PROGRAMS

PROBLEM STATEMENT 1:

Design YACC/LEX code for implementing simple Desk Calculator.

LEX FILE

```
%{
#include "y.tab.h"
%}
%%
[0-9]+      { yylval = atoi(yytext); return NUMBER; }
[\n]        { return '\n'; }
[ \t]       ;           // Ignore whitespace
.           { return yytext[0]; }  // Return character as token
%%
int yywrap() { return 1; }
```

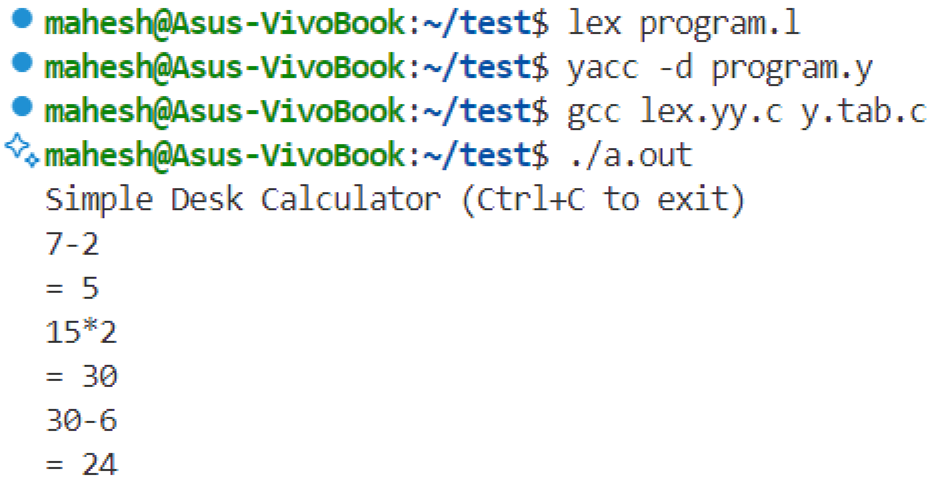
YACC FILE

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

void yyerror(const char *s);
int yylex();
%}

%token NUMBER
```

```
%left '+' '-'
%left '*' '/'
%left UMINUS

%%

input:
    | input line
    ;

line:
    '\n'
    | expr '\n'   { printf("= %d\n", $1); }
    ;

expr:
    expr '+' expr    { $$ = $1 + $3; }
  | expr '-' expr    { $$ = $1 - $3; }
  | expr '*' expr    { $$ = $1 * $3; }
  | expr '/' expr    {
                         if ($3 == 0) {
                             printf("Error: Division by zero\n");
                             $$ = 0;
                         } else $$ = $1 / $3;
                       }
  | '-' expr %prec UMINUS { $$ = -$2; }
  | '(' expr ')'      { $$ = $2; }
```

```
 | NUMBER          { $$ = $1; }
 ;
%%
void yyerror(const char *s) {
   printf("Error: %s\n", s);
}
int main() {
   printf("Simple Desk Calculator (Ctrl+C to exit)\n");
   return yyparse();
}
```

OUTPUT



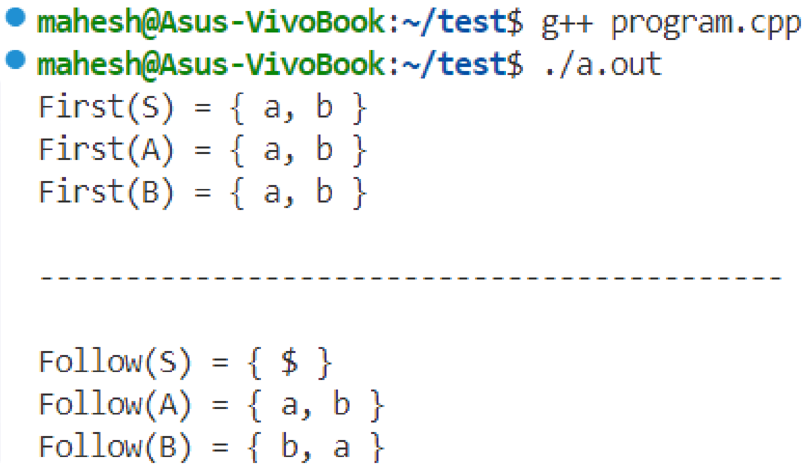
```
mahesh@Asus-VivoBook:~/test$ lex program.l
mahesh@Asus-VivoBook:~/test$ yacc -d program.y
mahesh@Asus-VivoBook:~/test$ gcc lex.yy.c y.tab.c
mahesh@Asus-VivoBook:~/test$ ./a.out
Simple Desk Calculator (Ctrl+C to exit)
7-2
= 5
15*2
= 30
30-6
= 24
```

PROBLEM STATEMENT 2

Program for computing FIRST and FOLLOW

```
#include <iostream>
#include <vector>
#include <string>
#include <cctype>
#include <algorithm>

using namespace std;

const int MAX = 10;
vector<string> productions = {
    "S=AaAb", "S=BbBa", "A=#", "B=#"
};
int countProductions = productions.size();

vector<vector<char>> calc_first(MAX);
vector<vector<char>> calc_follow(MAX);
vector<char> first;
vector<char> followSet;
char ck;

bool isPresent(const vector<char>& vec, char c) {
    return find(vec.begin(), vec.end(), c) != vec.end();
}

void findFirst(char c, int q1 = 0, int q2 = 0) {
    if (!isupper(c)) {
        first.push_back(c);
        return;
    }

    for (int j = 0; j < countProductions; j++) {
        if (productions[j][0] == c) {
            if (productions[j].size() <= 2) continue;

            if (productions[j][2] == '#') {
                if (q2 < productions[q1].size()) {
                    findFirst(productions[q1][q2], q1, q2 + 1);
                } else {
                    first.push_back('#');
                }
            } else if (!isupper(productions[j][2])) {
                first.push_back(productions[j][2]);
            } else {
                findFirst(productions[j][2], j, 3);
            }
        }
    }
}

void followFirst(char c, int c1, int c2);

void follow(char c) {
    if (productions[0][0] == c) {
        followSet.push_back('$');
    }

    for (int i = 0; i < countProductions; i++) {
        string prod = productions[i];
        for (int j = 2; j < prod.size(); j++) {
            if (prod[j] == c) {
                if (j + 1 < prod.size()) {
                    followFirst(prod[j + 1], i, j + 2);
                }
                if (j + 1 == prod.size() && prod[0] != c) {
                    follow(prod[0]);
                }
            }
        }
    }
}

void followFirst(char c, int c1, int c2) {
    if (!isupper(c)) {
        followSet.push_back(c);
    } else {
        for (int i = 0; i < countProductions; i++) {
            if (calc_first[i].size() > 0 && calc_first[i][0] == c) {
                for (int j = 1; j < calc_first[i].size(); j++) {
                    char sym = calc_first[i][j];
                    if (sym != '#' && !isPresent(followSet, sym)) {
                        followSet.push_back(sym);
                    } else if (sym == '#') {
                        if (c2 < productions[c1].size()) {
                            followFirst(productions[c1][c2], c1, c2 + 1);
                        } else {
                            follow(productions[c1][0]);
                        }
                    }
                }
                break;
            }
        }
    }
}

int main() {
    vector<char> done;

    // FIRST
    for (int k = 0; k < countProductions; k++) {
        char c = productions[k][0];
        if (!isPresent(done, c)) {
            first.clear();
            findFirst(c, 0, 0);
            done.push_back(c);
            vector<char> result = {c};
            for (char ch : first) {
                if (!isPresent(result, ch))
                    result.push_back(ch);
            }
            calc_first[k] = result;
            cout << "First(" << c << ") = { ";
            for (int i = 1; i < result.size(); i++)
                cout << result[i] << (i < result.size() - 1 ? ", " : "");
            cout << " }" << endl;
        }
    }

    cout << "\n----------------------------------------\n\n";

    // FOLLOW
    done.clear();
    for (int k = 0; k < countProductions; k++) {
        char c = productions[k][0];
        if (!isPresent(done, c)) {
            followSet.clear();
            ck = c;
            follow(c);
            done.push_back(c);
            vector<char> result = {c};
            for (char ch : followSet) {
                if (!isPresent(result, ch))
                    result.push_back(ch);
            }
            calc_follow[k] = result;
            cout << "Follow(" << c << ") = { ";
            for (int i = 1; i < result.size(); i++)
                cout << result[i] << (i < result.size() - 1 ? ", " : "");
            cout << " }" << endl;
        }
    }

    return 0;
}
```

OUTPUT



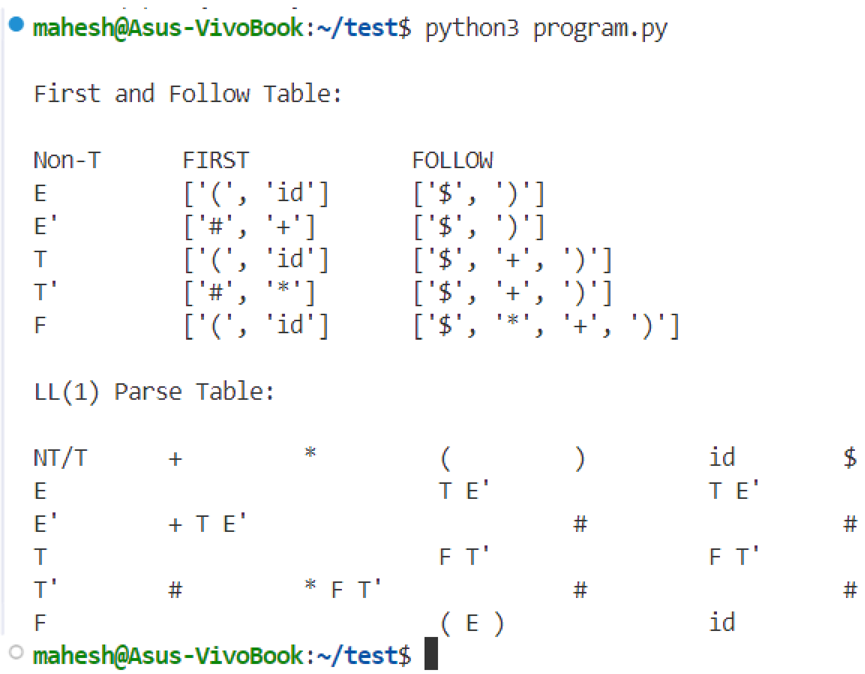
```
● mahesh@Asus-VivoBook:~/test$ g++ program.cpp
● mahesh@Asus-VivoBook:~/test$ ./a.out
  First(S) = { a, b }
  First(A) = { a, b }
  First(B) = { a, b }

  ------------------------------------------

  Follow(S) = { $ }
  Follow(A) = { a, b }
  Follow(B) = { b, a }
```

PROBLEM STATEMENT 3

Program for LL(1) parsing table.

```
import copy
term_userdef = []
nonterm_userdef = []
diction = {}
firsts = {}
follows = {}
start_symbol = ""


def removeLeftRecursion(rules):
    new_rules = []
    for rule in rules:
        lhs, rhs = rule.split("->")
        lhs = lhs.strip()
        alphas = []
        betas = []
        for prod in rhs.split('|'):
            prod = prod.strip().split()
            if prod[0] == lhs:
                alphas.append(prod[1:])
            else:
                betas.append(prod)
        if alphas:
            new_lhs = lhs + "'"
            nonterm_userdef.append(new_lhs)
            diction[lhs] = []
            for beta in betas:
                diction[lhs].append(beta + [new_lhs])
            diction[new_lhs] = []
            for alpha in alphas:
                diction[new_lhs].append(alpha + [new_lhs])
            diction[new_lhs].append(['#'])
        else:
            diction[lhs] = [prod.strip().split() for prod in rhs.split('|')]
    return diction


def leftFactoring():
    for lhs in list(diction.keys()):
        productions = diction[lhs]
        prefix_map = {}
        for prod in productions:
            prefix = prod[0]
            prefix_map.setdefault(prefix, []).append(prod)

        new_productions = []
        for prefix, group in prefix_map.items():
            if len(group) > 1:
                new_nt = lhs + "'"
                i = 1
                while new_nt in diction or new_nt in nonterm_userdef:
                    new_nt += "'"
                nonterm_userdef.append(new_nt)
                new_group = [p[1:] if len(p) > 1 else ['#'] for p in group]
                diction[new_nt] = new_group
                new_productions.append([prefix, new_nt])
            else:
                new_productions.append(group[0])
        diction[lhs] = new_productions

def first(symbol):
    if symbol in term_userdef:
        return [symbol]
    if symbol == '#':
        return ['#']
    if symbol not in diction:
        return []

    result = []
    for production in diction[symbol]:
        for sym in production:
            temp = first(sym)
            result += [t for t in temp if t != '#']
            if '#' not in temp:
                break
        else:
            result.append('#')
    return list(set(result))

def follow(symbol):
    result = []
    if symbol == start_symbol:
        result.append('$')
    for lhs in diction:
        for production in diction[lhs]:
            for i, sym in enumerate(production):
                if sym == symbol:
                    if i + 1 < len(production):
                        next_sym = production[i + 1]
                        first_next = first(next_sym)
                        result += [f for f in first_next if f != '#']
                        if '#' in first_next:
                            result += follow(lhs)
                    else:
                        if lhs != symbol:
                            result += follow(lhs)
    return list(set(result))


def computeAllFirsts():
    for nt in nonterm_userdef:
        firsts[nt] = first(nt)


def computeAllFollows():
    for nt in nonterm_userdef:
        follows[nt] = follow(nt)


def createParseTable():
    print("\nFirst and Follow Table:\n")
    mx_len_first = max(len(str(firsts[u])) for u in diction)
    mx_len_fol = max(len(str(follows[u])) for u in diction)

    print(f"{'Non-T':<10} {'FIRST':<{mx_len_first + 5}} {'FOLLOW':<{mx_len_fol + 5}}")
    for u in diction:
        print(f"{u:<10} {str(firsts[u]):<{mx_len_first + 5}} {str(follows[u]):<{mx_len_fol +
5}}")

    parse_table = {}
    for nt in diction:
        parse_table[nt] = {}
        for prod in diction[nt]:
            first_prod = first(prod[0])
            for terminal in first_prod:
                if terminal != '#':
                    parse_table[nt][terminal] = prod
            if '#' in first_prod:
                for follow_sym in follows[nt]:
                    parse_table[nt][follow_sym] = ['#']

    print("\nLL(1) Parse Table:\n")
    terminals = list(term_userdef) + ['$']
    header = ['NT/T'] + terminals
    print("{:<10}".format(header[0]), end='')
    for t in header[1:]:
        print("{:<10}".format(t), end='')
    print()

    for nt in parse_table:
        print("{:<10}".format(nt), end='')
        for t in terminals:
            rule = parse_table[nt].get(t, '')
            rule_str = ' '.join(rule) if rule else ''
            print("{:<10}".format(rule_str), end='')
        print()
    return parse_table

rules = [
    "E -> T E'",
    "E' -> + T E' | #",
    "T -> F T'",
    "T' -> * F T' | #",
    "F -> ( E ) | id"
]

nonterm_userdef = ['E', "E'", 'T', "T'", 'F']
term_userdef = ['+', '*', '(', ')', 'id']

start_symbol = 'E'

removeLeftRecursion(rules)
leftFactoring()
computeAllFirsts()
```

```
computeAllFollows()

parse_table = createParseTable()
```



```
mahesh@Asus-VivoBook:~/test$ python3 program.py

First and Follow Table:

Non-T      FIRST            FOLLOW
E          ['(', 'id']      ['$', ')']
E'         ['#', '+']       ['$', ')']
T          ['(', 'id']      ['$', '+', ')']
T'         ['#', '*']       ['$', '+', ')']
F          ['(', 'id']      ['$', '*', '+', ')']

LL(1) Parse Table:

NT/T      +         *         (         )         id        $
E                             T E'                T E'
E'        + T E'                        #                   #
T                             F T'                F T'
T'        #         * F T'              #                   #
F                             ( E )               id
mahesh@Asus-VivoBook:~/test$ 
```

PROBLEM STATEMENT 4

Design YACC/LEX code for generating 3AC for simple arithmetic expressions.

LEX FILE

```
%{
#include "y.tab.h"
#include <string.h>
%}

%%
[0-9]+                { yylval.str = strdup(yytext); return NUMBER; }
[a-zA-Z_][a-zA-Z0-9_]*  { yylval.str = strdup(yytext); return ID; }
[+\-*/()]             { return yytext[0]; }
[ \t\n]+              { /* skip whitespace */ }
.                     { printf("Invalid character: %s\n", yytext); }

%%
int yywrap() {
   return 1;
}
```

YACC FILE

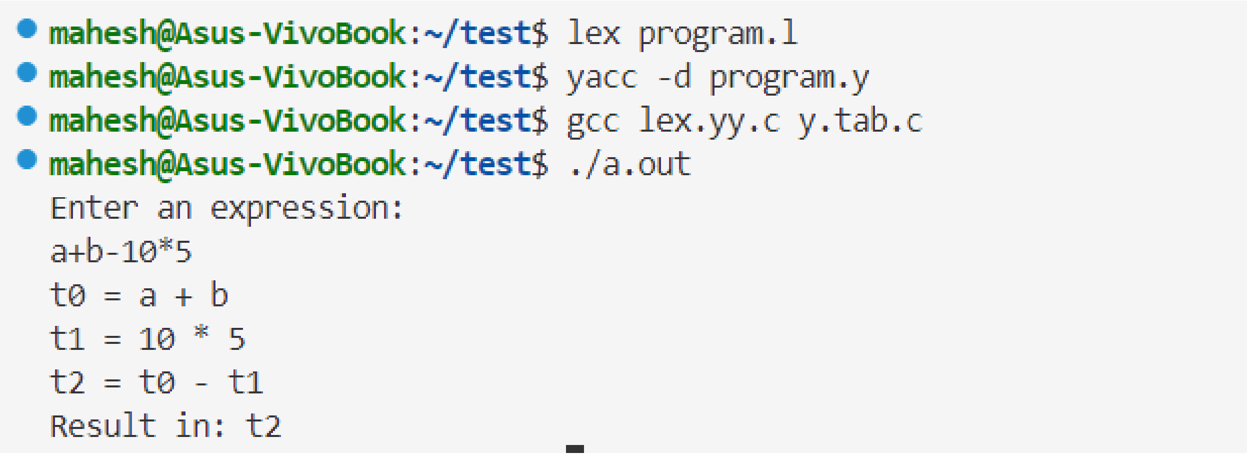
```
%{
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
// Temporary variable count for generating new temporaries
int temp_count = 0;

// Function to create new temporary variable name
char* new_temp() {
    char* temp = (char*)malloc(10);
    sprintf(temp, "t%d", temp_count++);
    return temp;
}

void yyerror(const char* s);
int yylex(void);

%}

// Declare semantic value type
%union {
    char* str;
}

%token <str> ID NUMBER
%type <str> expr

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

%%
```

```
stmt:
  expr {
    printf("Result in: %s\n", $1);
    free($1);
  }
;

expr:
   expr '+' expr {
     char* temp = new_temp();
     printf("%s = %s + %s\n", temp, $1, $3);
     free($1);
     free($3);
     $$ = temp;
   }
  | expr '-' expr {
     char* temp = new_temp();
     printf("%s = %s - %s\n", temp, $1, $3);
     free($1);
     free($3);
     $$ = temp;
   }
  | expr '*' expr {
     char* temp = new_temp();
     printf("%s = %s * %s\n", temp, $1, $3);
     free($1);
```

```
        free($3);
        $$ = temp;
      }
    | expr '/' expr {
        char* temp = new_temp();
        printf("%s = %s / %s\n", temp, $1, $3);
        free($1);
        free($3);
        $$ = temp;
      }
    | '(' expr ')' {
        $$ = $2;
      }
    | ID {
        $$ = strdup($1);
        free($1);
      }
    | NUMBER {
        $$ = strdup($1);
        free($1);
      }
;
%%

int main() {
    printf("Enter an expression:\n");
    return yyparse();
```

```
}


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

OUTPUT



```
● mahesh@Asus-VivoBook:~/test$ lex program.l
● mahesh@Asus-VivoBook:~/test$ yacc -d program.y
● mahesh@Asus-VivoBook:~/test$ gcc lex.yy.c y.tab.c
● mahesh@Asus-VivoBook:~/test$ ./a.out
  Enter an expression:
  a+b-10*5
  t0 = a + b
  t1 = 10 * 5
  t2 = t0 - t1
  Result in: t2
```

PROBLEM STATEMENT 5

Design YACC/LEX code for constructing Syntax tree for simple arithmetic expressions.

node.h FILE

```
#ifndef NODE_H
#define NODE_H
typedef struct Node {
    char* value;
    struct Node* left;
    struct Node* right;
} Node;
#endif
```

LEX FILE

```
%{
#include "y.tab.h"
#include <string.h>
%}

%%
[0-9]+                  { yylval.str = strdup(yytext); return NUMBER; }
[a-zA-Z_][a-zA-Z0-9_]*  { yylval.str = strdup(yytext); return ID; }
[+\-*/()]               { return yytext[0]; }
[ \t\n]+                ;  /* ignore whitespace */
.                       { printf("Invalid character: %s\n", yytext); }
%%
int yywrap(){
    return 1;
}
```

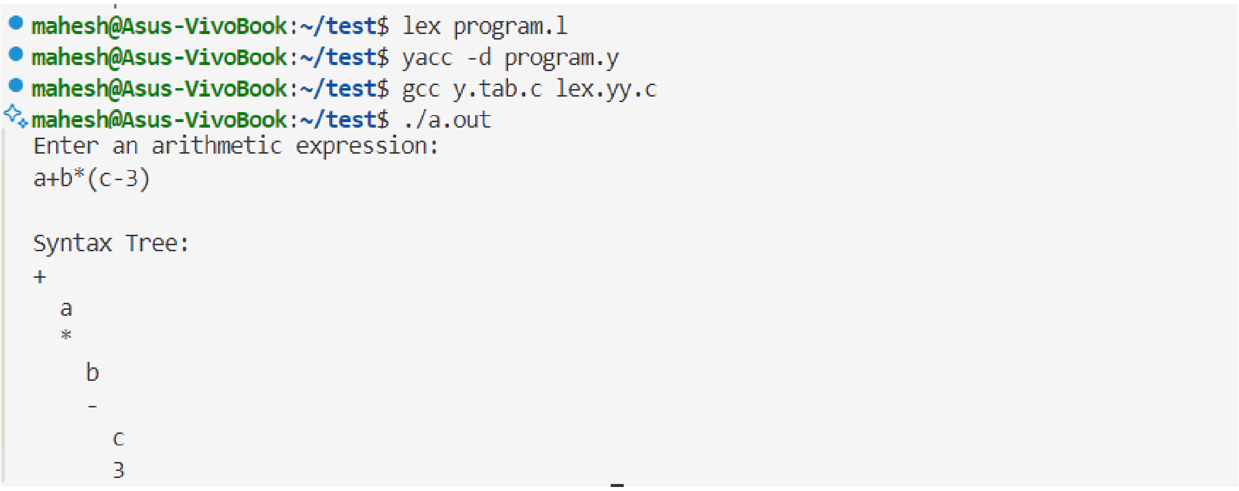
YACC FILE

```
%{
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include "node.h"

Node* create_node(char* value, Node* left, Node* right);
void print_tree(Node* node, int level);
void free_tree(Node* node);

int yylex(void);
void yyerror(const char* s);
%}

%union {
    char* str;
    Node* node;
}

%token <str> ID NUMBER
%type <node> expr

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

```
%%

stmt:
    expr {
        printf("\nSyntax Tree:\n");
        print_tree($1, 0);
        free_tree($1);
    }
;

expr:
      expr '+' expr { $$ = create_node("+", $1, $3); }
    | expr '-' expr { $$ = create_node("-", $1, $3); }
    | expr '*' expr { $$ = create_node("*", $1, $3); }
    | expr '/' expr { $$ = create_node("/", $1, $3); }
    | '(' expr ')'  { $$ = $2; }
    | ID            { $$ = create_node($1, NULL, NULL); free($1); }
    | NUMBER        { $$ = create_node($1, NULL, NULL); free($1); }
;

%%

Node* create_node(char* value, Node* left, Node* right) {
    Node* node = (Node*)malloc(sizeof(Node));
    node->value = strdup(value);
    node->left = left;
    node->right = right;
    return node;
}

void print_tree(Node* node, int level) {
    if (!node) return;
    for (int i = 0; i < level; i++) printf("  ");
    printf("%s\n", node->value);
    print_tree(node->left, level + 1);
    print_tree(node->right, level + 1);
}

void free_tree(Node* node) {
    if (!node) return;
    free_tree(node->left);
    free_tree(node->right);
    free(node->value);
    free(node);
}

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

int main() {
    printf("Enter an arithmetic expression:\n");
    return yyparse();
```

}

OUTPUT



```
mahesh@Asus-VivoBook:~/test$ lex program.l
mahesh@Asus-VivoBook:~/test$ yacc -d program.y
mahesh@Asus-VivoBook:~/test$ gcc y.tab.c lex.yy.c
mahesh@Asus-VivoBook:~/test$ ./a.out
Enter an arithmetic expression:
a+b*(c-3)

Syntax Tree:
+
  a
  *
    b
    -
      c
      3
```

PROBLEM STATEMENT 6

Design YACC/ LEX code to convert infix expression to postfix expression.

LEX FILE

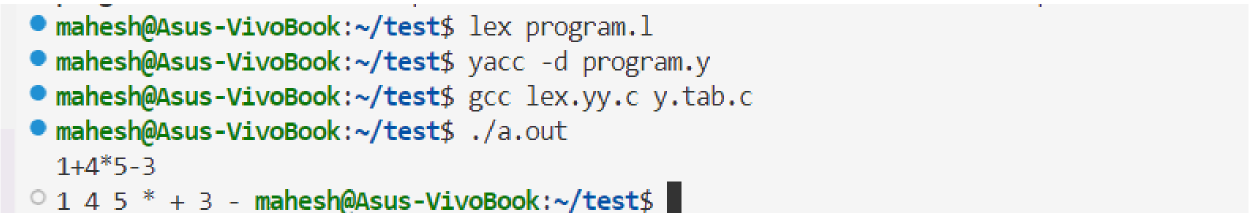
```
%{
#include "y.tab.h"
#include <stdlib.h>
extern int yylval;
int yywrap();
%}
%%
[0-9]       { yylval = atoi(yytext); return ID; }
[\n]        { return '\n'; }
[+\-*/^()]         { return yytext[0]; }

%%
int yywrap() { return 1; }
```

YACC FILE

```
%{
#include<stdio.h>
int yylex(void);
void yyerror(const char *s);
%}
%start S
%token ID
%left '+' '-'
```

```
%left '*' '/'
%right '^'


%%
S:E '\n' ;
E:E '+' E {printf("+ ");}|
E '-' E {printf("- ");}|
E '*' E {printf("* ");}|
E '/' E {printf("/ ");}|
E '^'  E {printf("^ ");}|
'(' E ')'{}|
ID {printf("%d ",$1);} ;
%%
int main(){
   yyparse();
   return 0;
}
void yyerror(const char *s){
   fprintf(stderr,"error not valid expression %s",s);
}
```

OUTPUT



```
mahesh@Asus-VivoBook:~/test$ lex program.l
mahesh@Asus-VivoBook:~/test$ yacc -d program.y
mahesh@Asus-VivoBook:~/test$ gcc lex.yy.c y.tab.c
mahesh@Asus-VivoBook:~/test$ ./a.out
1+4*5-3
1 4 5 * + 3 - mahesh@Asus-VivoBook:~/test$
```