VISVESVARAYA TECHNOLOGICAL UNIVERSITY

"JnanaSangama", Belgaum -590014, Karnataka.

LAB REPORT

on

COMPILER DESIGN

Submitted by

Harsh Kumar (1BM21CS261)

Under the Guidance of

Prof. Latha NR
Assistant Professor,
BMSCE

in partial fulfilment for the award of the degree of

BACHELOR OF ENGINEERING

in

COMPUTER SCIENCE AND ENGINEERING

B.M.S. COLLEGE OF ENGINEERING

(Autonomous Institution under VTU)

BENGALURU-560019

November 2023-February 2024

B. M. S. College of Engineering,

Bull Temple Road, Bangalore 560019

(Affiliated To Visvesvaraya Technological University, Belgaum)

Department of Computer Science and Engineering

CERTIFICATE

This is to certify that the Lab work entitled "**Compiler Design**" carried out by **Harsh Kumar (1BM21CS261)** , who is bonafide student of **B. M. S. College of Engineering.** It is in partial fulfilment for the award of **Bachelor of Engineering in Computer Science and Engineering** of the Visvesvaraya Technological University, Belgaum during the year 2023-24.

The Lab report has been approved as it satisfies the academic requirements in respect of **Compiler Design- (22CS5PCCPD)** work prescribed for the said degree.

Prof. Latha NR	Dr. Jyothi Nayak
Assistant professor	Professor and Head
Department of CSE	Department of CSE
BMSCE, Bengaluru	BMSCE, Bengaluru

B. M. S. COLLEGE OF ENGINEERING

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

DECLARATION

I, Harsh Kumar (1BM21CS261), student of 5th Semester, B.E, Department of Computer Science and Engineering, B. M. S. College of Engineering, Bangalore, here by declare that, this lab report entitled " **Compiler Design**" has been carried out by me under theguidance of Prof. Latha NR, Assistant Professor, Department of CSE, B. M. S. College of Engineering, Bangalore during the academic semester November-2023-February-2024.

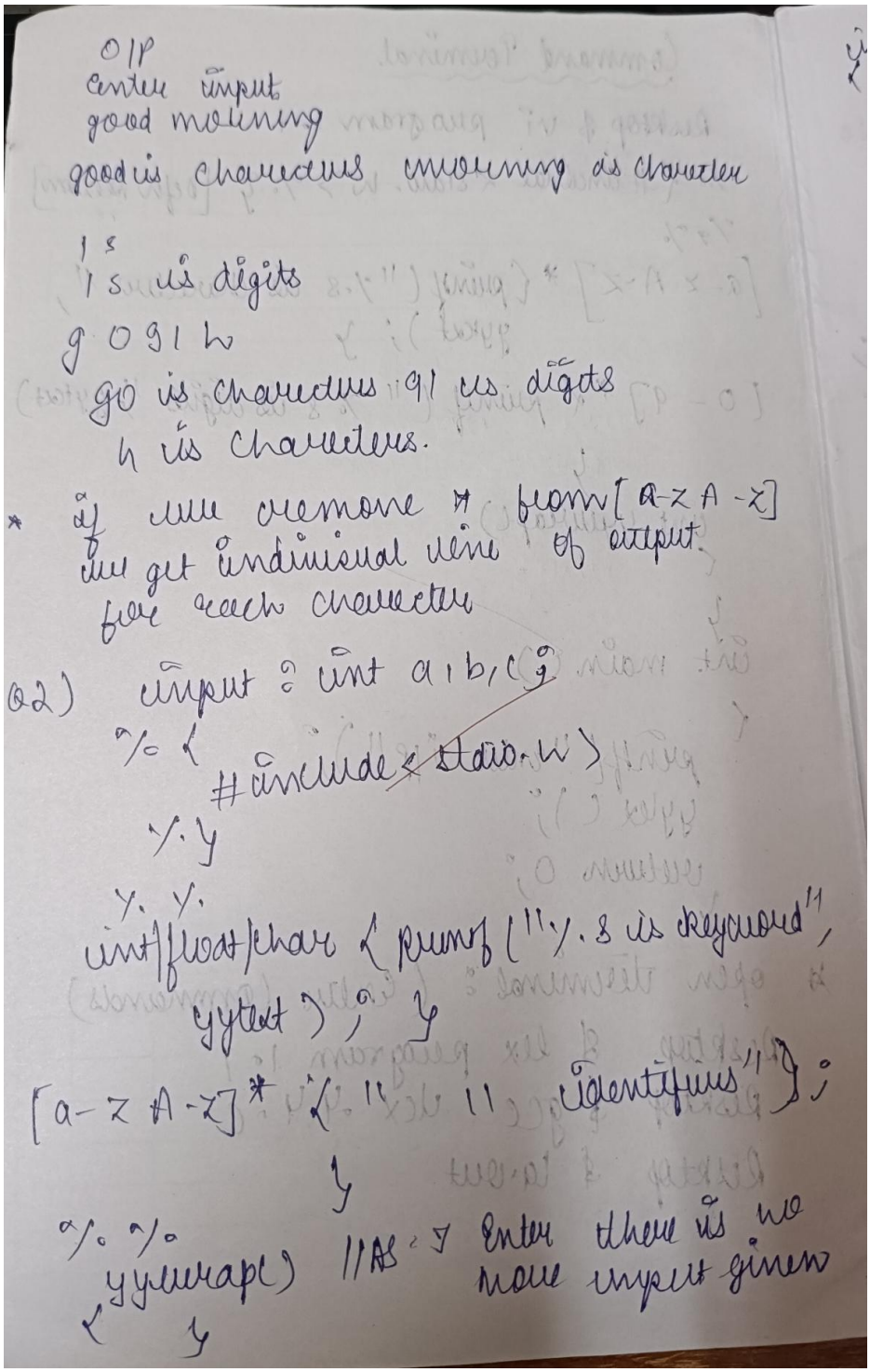
I also declare that to the best of my knowledge and belief, the development reported here is not from part of any other report by any other students.

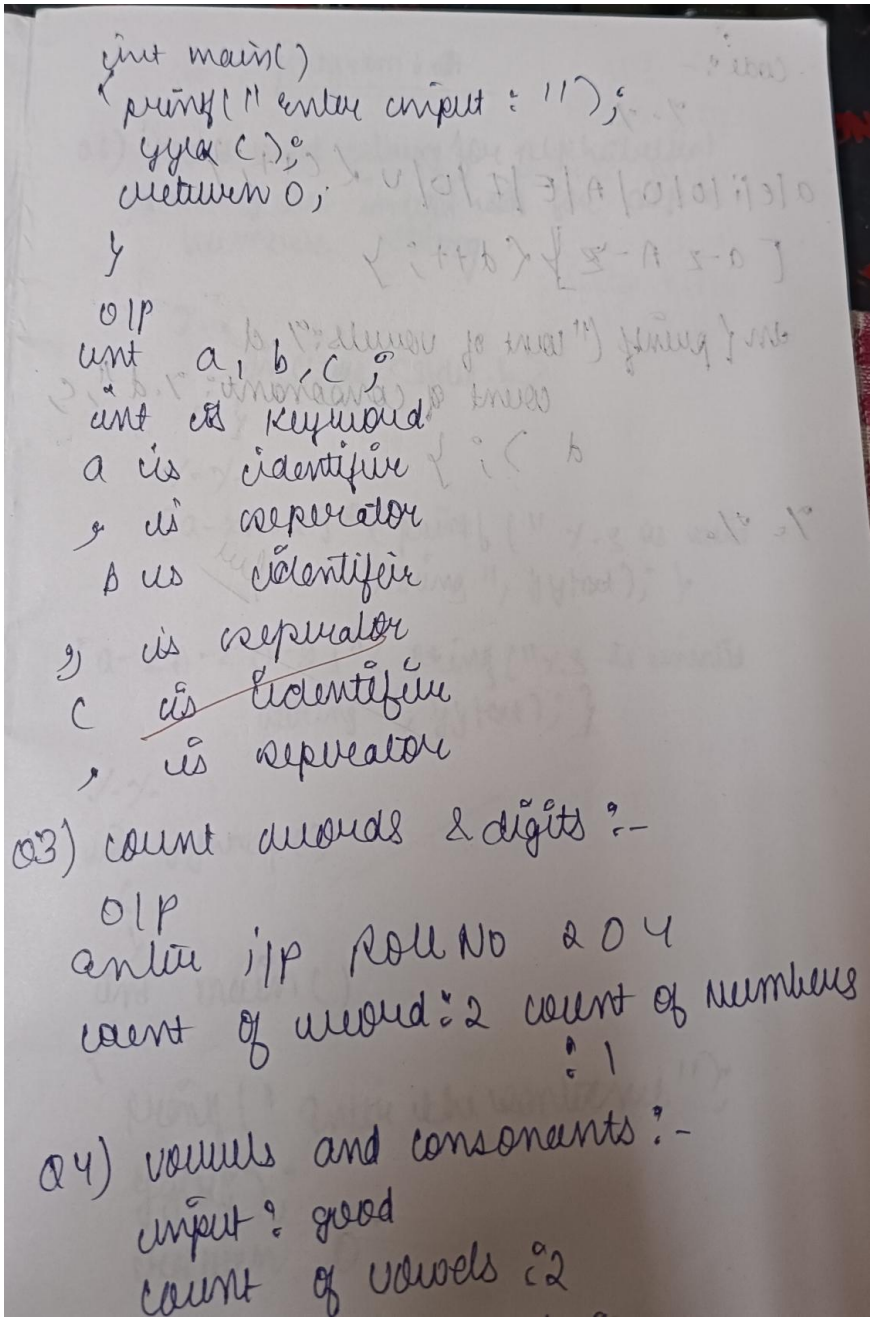
TABLE OF CONTENTS

Lab 1

1.1 Write a program in LEX to recognize different tokens: Keywords, Identifiers, Constants, Operators and Punctuation symbols.

Code:



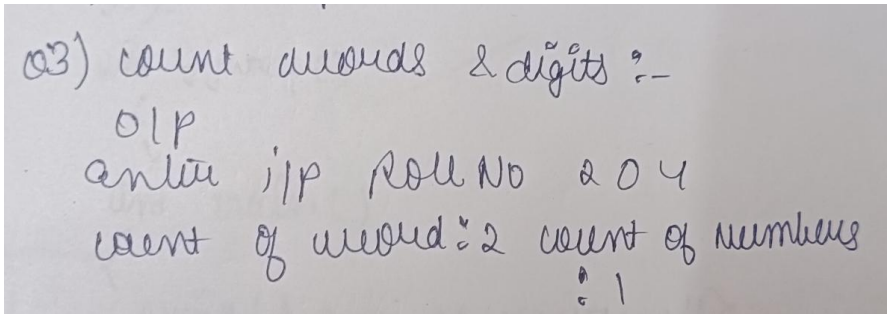


Output

```
Give an input:
int sum,x=2,y=3,z;
int-keyword
 sum-Identifier
,-separator
x-Identifier
=-assignment operator
2-digit
,-separator
y-Identifier
=-assignment operator
3-digit
,-separator
z-Identifier
;-delimiter
```

1.2 Write a program in LEX to count the number of characters and digits in a string.

Code

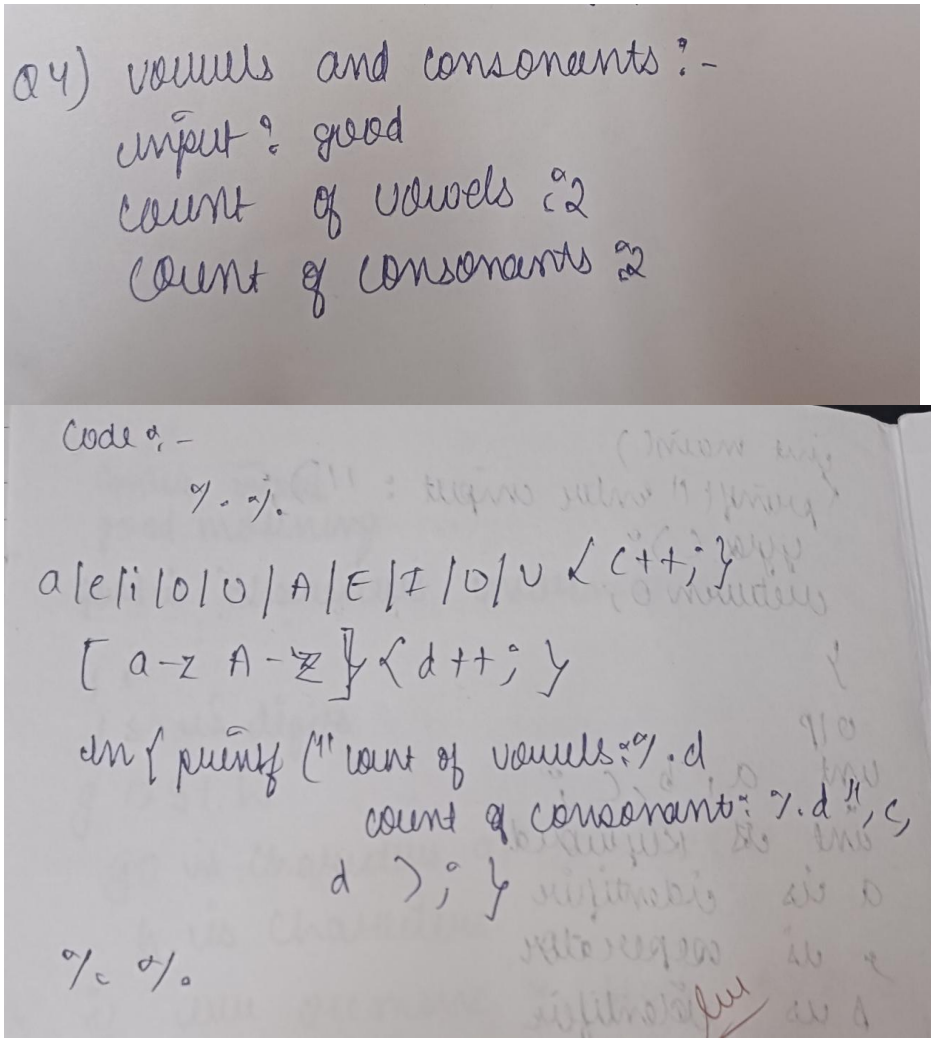


Output

```
Enter a sentence:
I was born in 2003.
No of characters and digits are 10 and 4
Hello123
No of characters and digits are 5 and 3
```

1.3 Write a program in LEX to count the number of vowels and consonants in a string.

Code



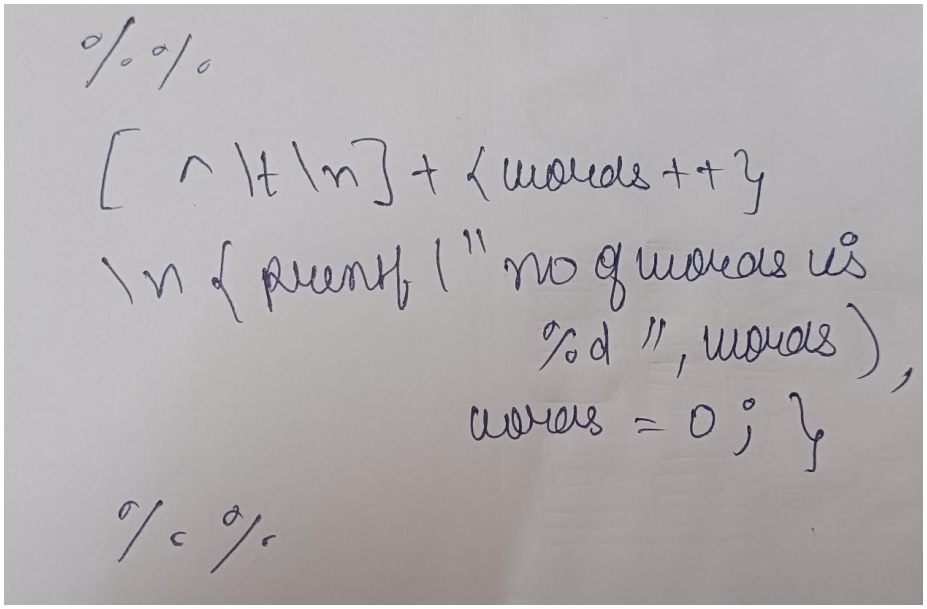
Output

```
Enter a sentence:
Compiler design
 No of vowels and consonants are 5 and 9
This is a book
   No of vowels and consonants are 5 and 6
```

Lab 2

2.1 Write a program in lex to count the number of words in a sentence.

Code

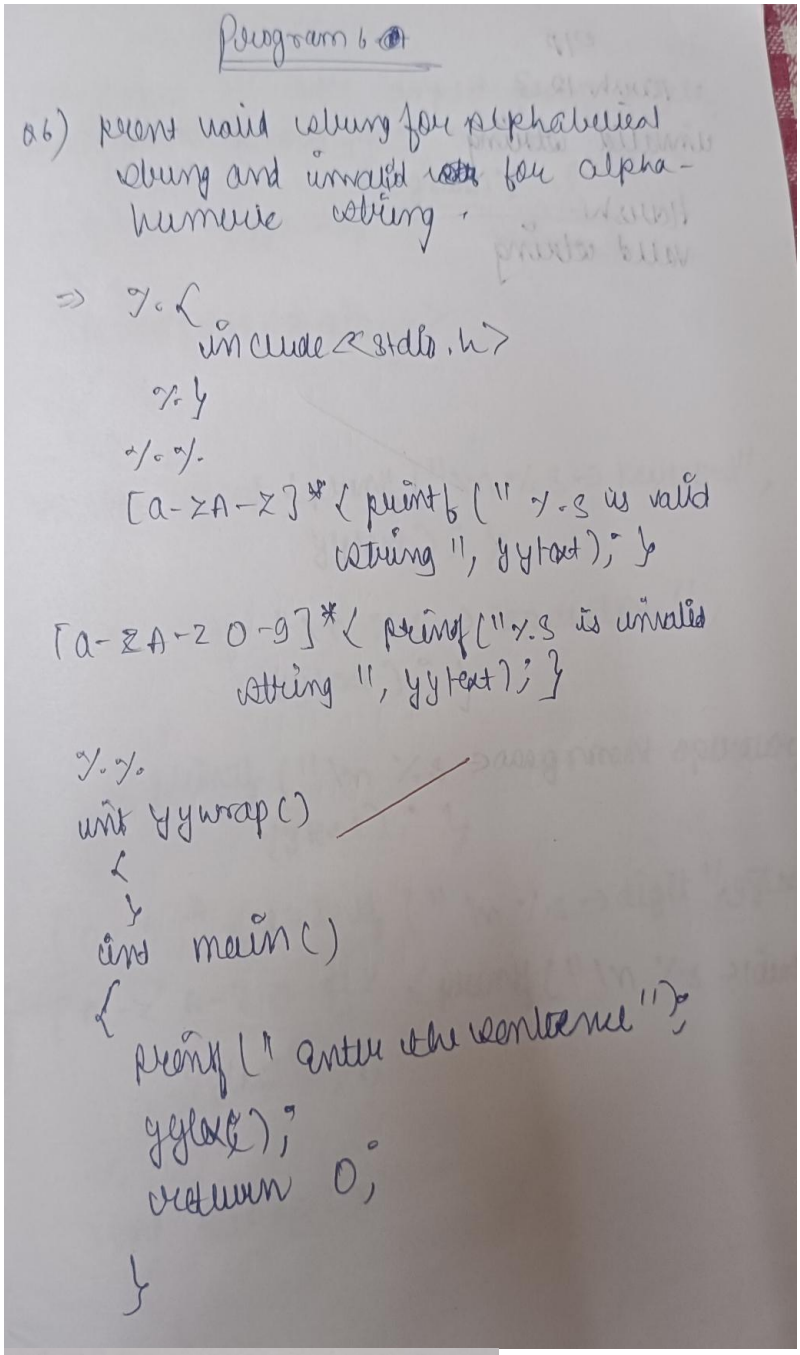


Output

```
Enter a sentence:
My name is Neha
   No of words in the sentence are 4.
I will make things happen.
    No of words in the sentence are 5.
```

2.2 Write a program in lex to demonstrate regular definition.

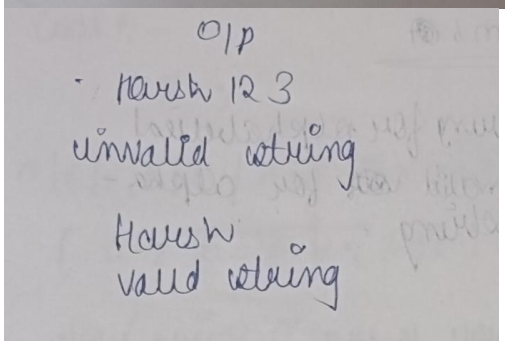
Code



Program 6

a6) Print valid string for alphabetical string and invalid for alpha-numeric string.

```
⇒ %{
    #include<stdio.h>
  %}
  %%
  [a-zA-Z]*{ printf("%s is valid string", yytext); }
  [a-zA-Z 0-9]*{ printf("%s is invalid string", yytext); }
  %%
  int yywrap()
  {
  }
  int main()
  {
    printf("enter the sentence");
    yylex();
    return 0;
  }
```



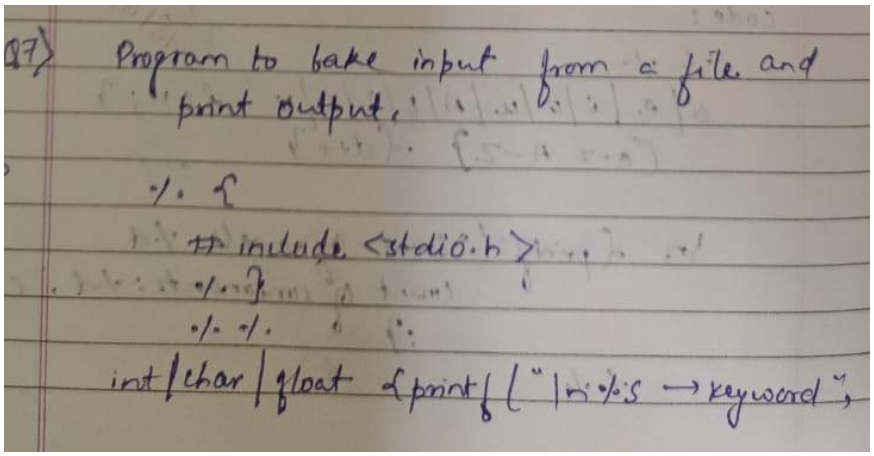
```
o/p
- Harsh 123
invalid string
Harsh
valid string
```

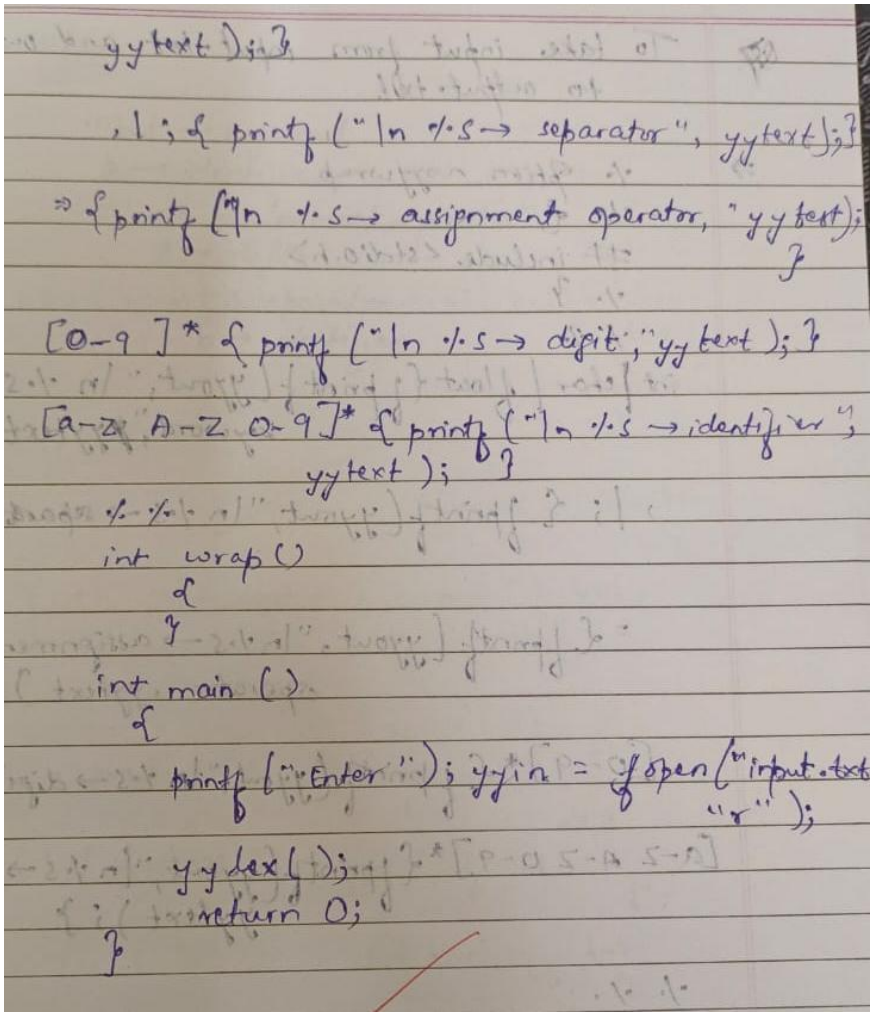
Output


Enter a string:
HelloWorld
Characters
1234
Digits
Hello123
Invalid input!

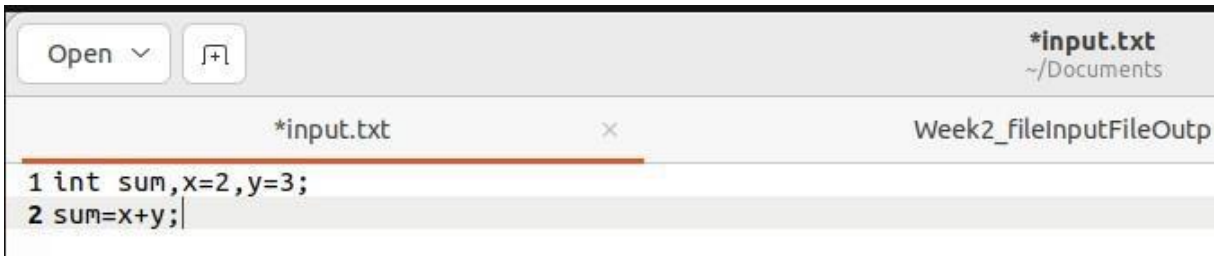
2.3 Write a program in lex to identify tokens in a program by taking input from a file and printing the output on the terminal.

Code





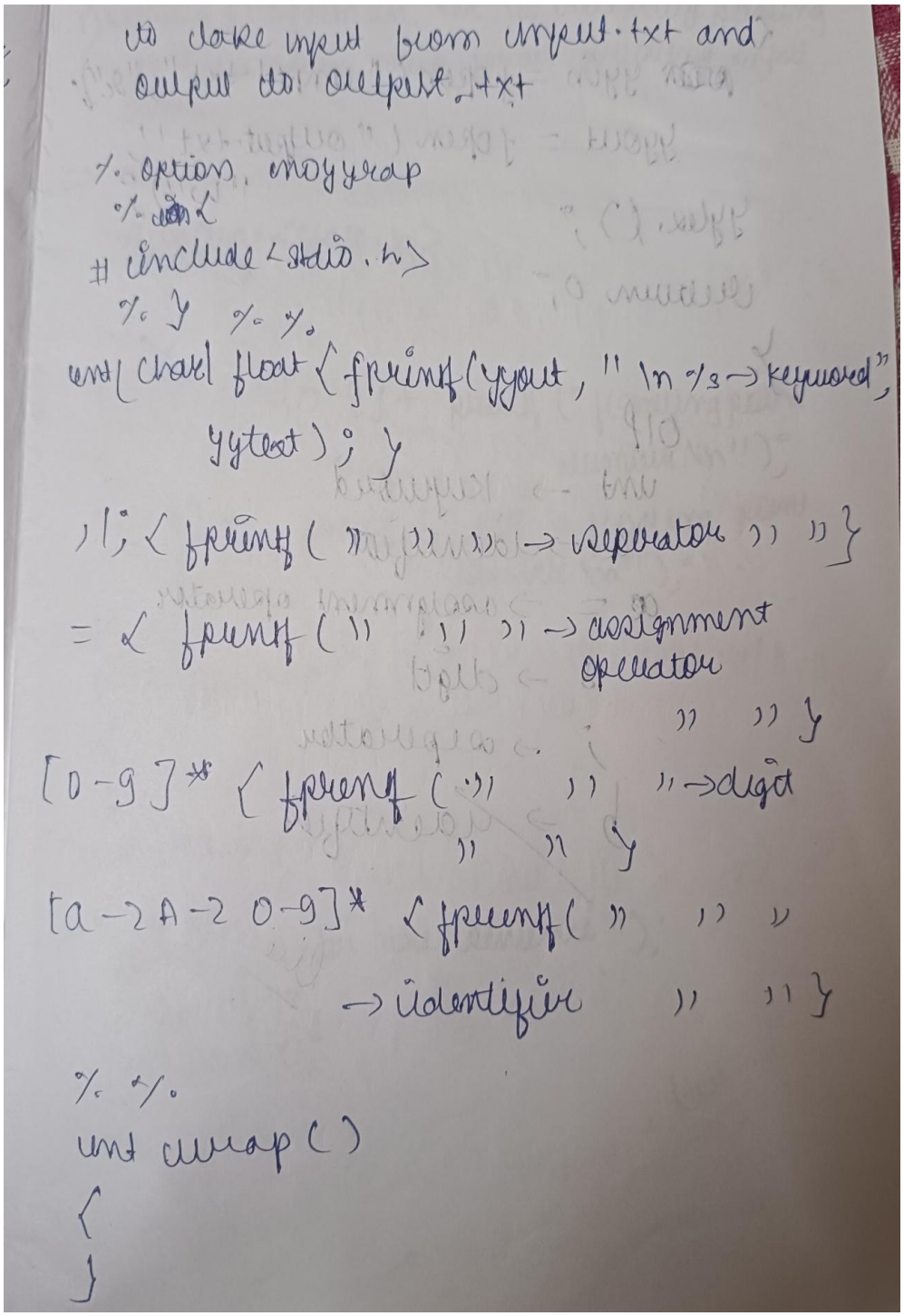
Output

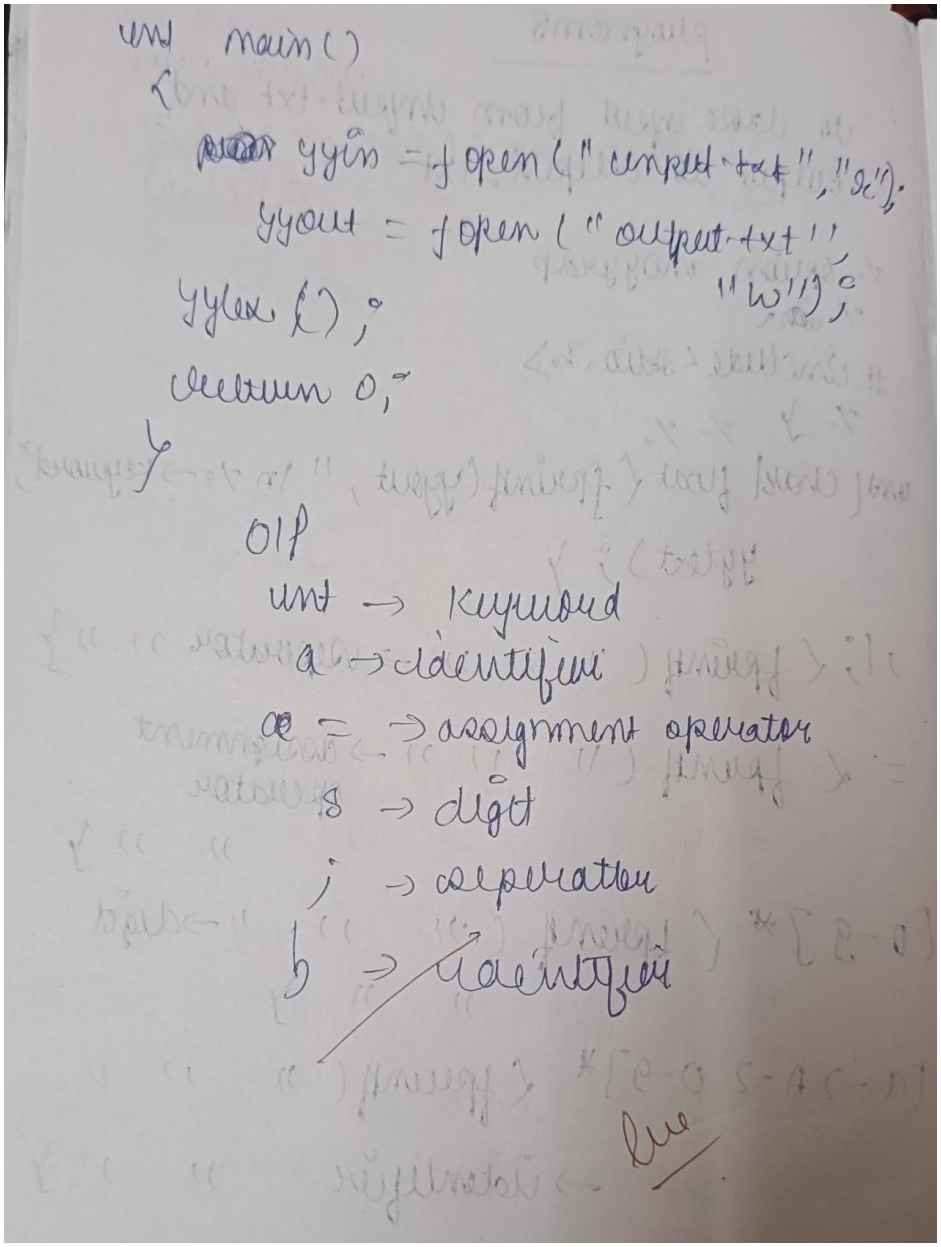



```
int is a keyword.
 sum is an identifier.
, is a separator.
x is an identifier.
= is an assignment operator.
2 is/are digit(s).
, is a separator.
y is an identifier.
= is an assignment operator.
3 is/are digit(s).
; is a delimiter.
sum is an identifier.
= is an assignment operator.
x is an identifier.
+ is a binary operator.
y is an identifier.
; is a delimiter.
```

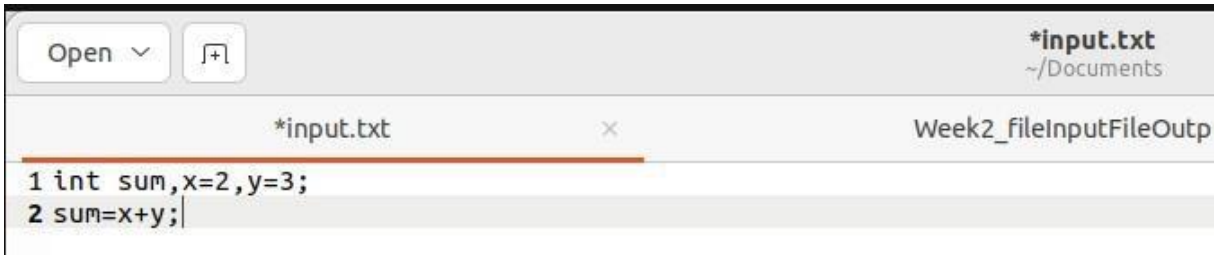
2.4 Write a program in lex to identify tokens in a program by taking input from a file and printing the output in another file.

Code

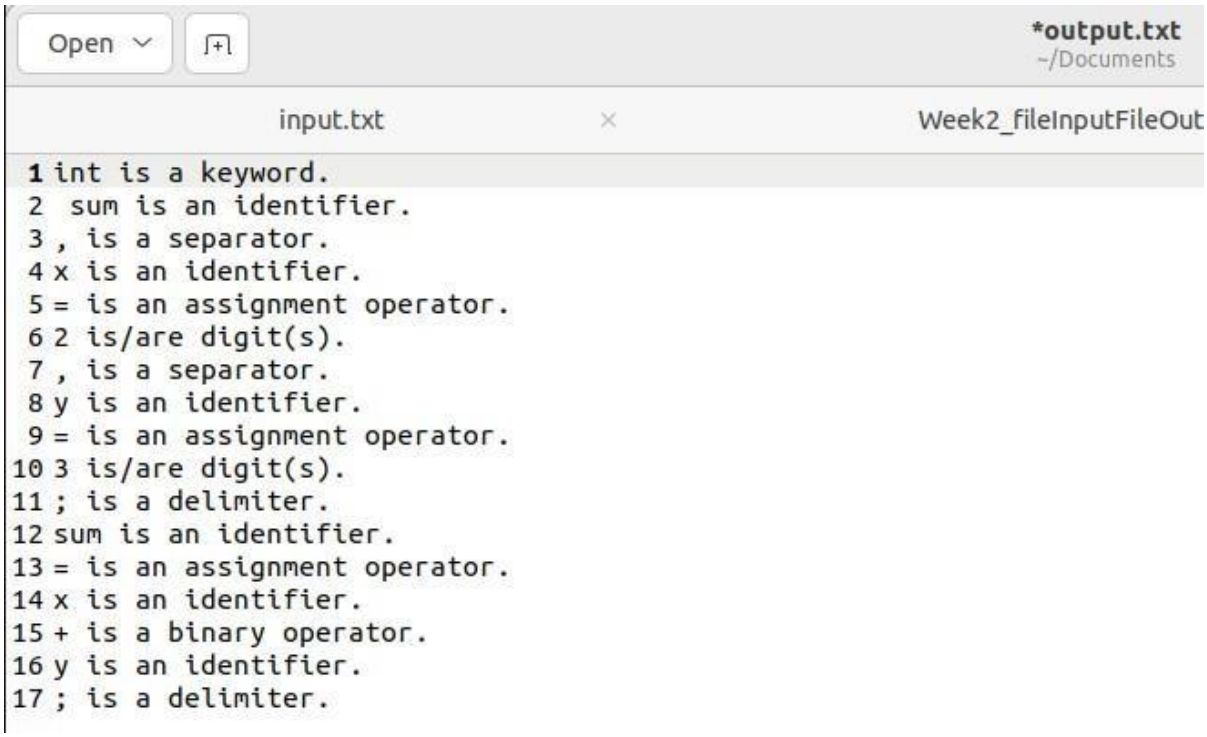




Output

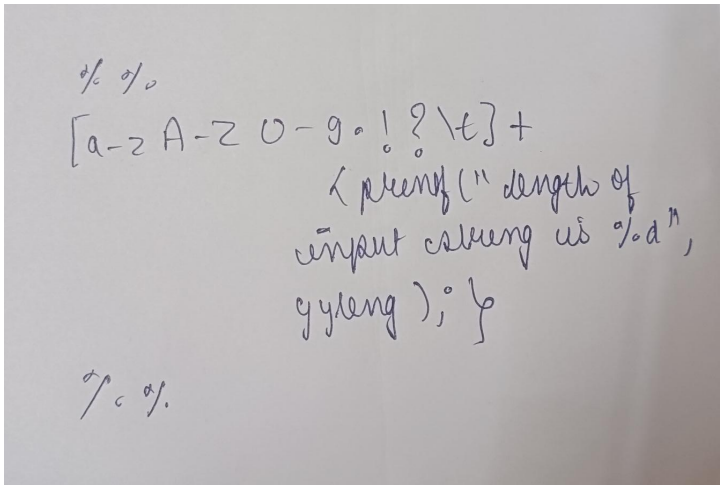



```
Printed in output.txt
```


Open
*output.txt
~/Documents
input.txt
Week2_fileInputFileOut
1 int is a keyword.
2 sum is an identifier.
3 , is a separator.
4 x is an identifier.
5 = is an assignment operator.
6 2 is/are digit(s).
7 , is a separator.
8 y is an identifier.
9 = is an assignment operator.
10 3 is/are digit(s).
11 ; is a delimiter.
12 sum is an identifier.
13 = is an assignment operator.
14 x is an identifier.
15 + is a binary operator.
16 y is an identifier.
17 ; is a delimiter.

2.5 Write a program in lex to find the length of the input string.

Code



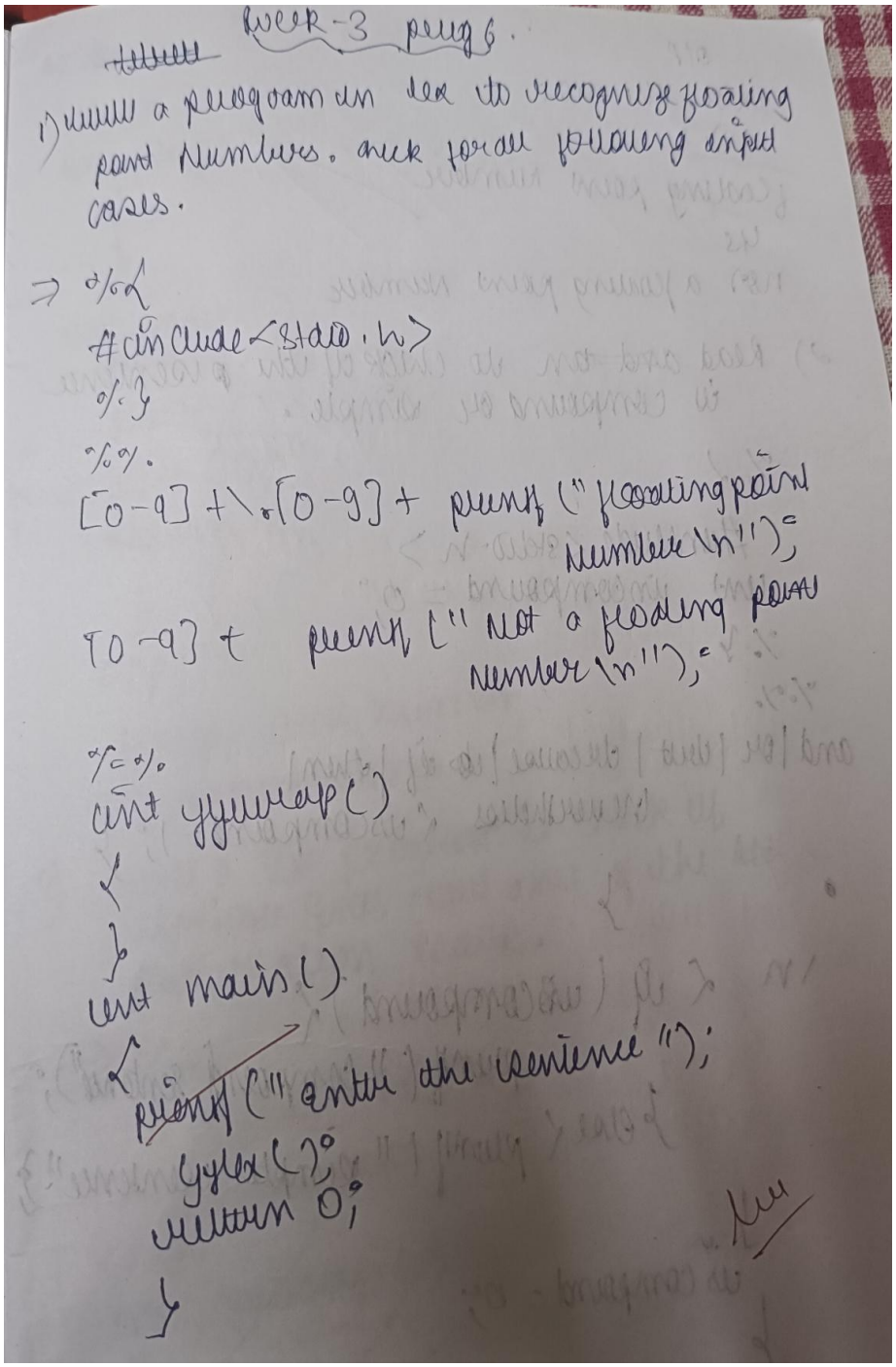
Output

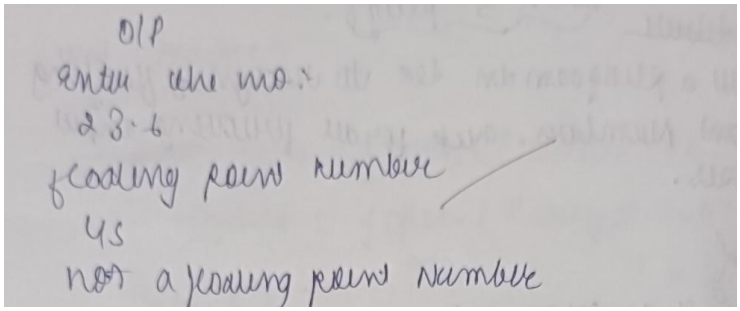
```
Enter a string:
Good Morning!
Length of input string is 13.

Where do you stay?
Length of input string is 18.
```

Lab 3

3.1 Write a program in LEX to recognize Floating Point Numbers.

Code



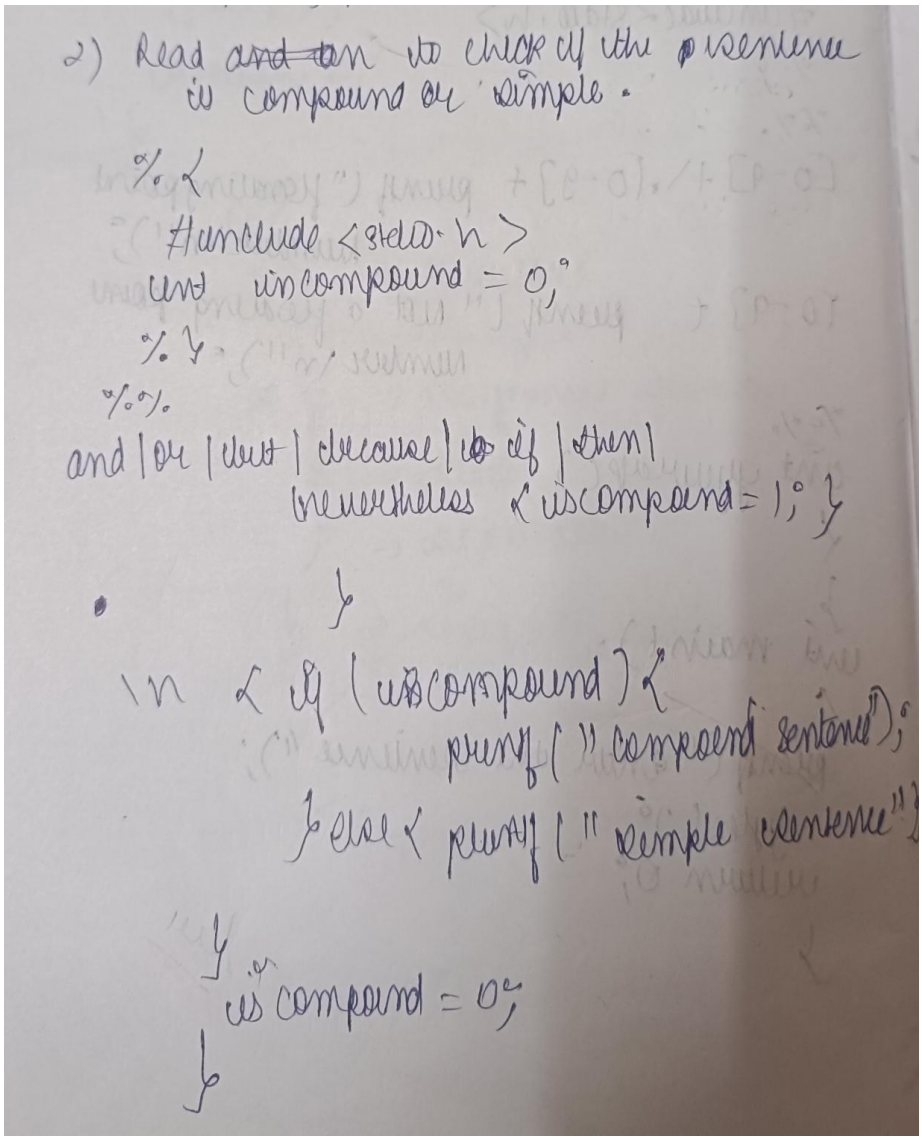


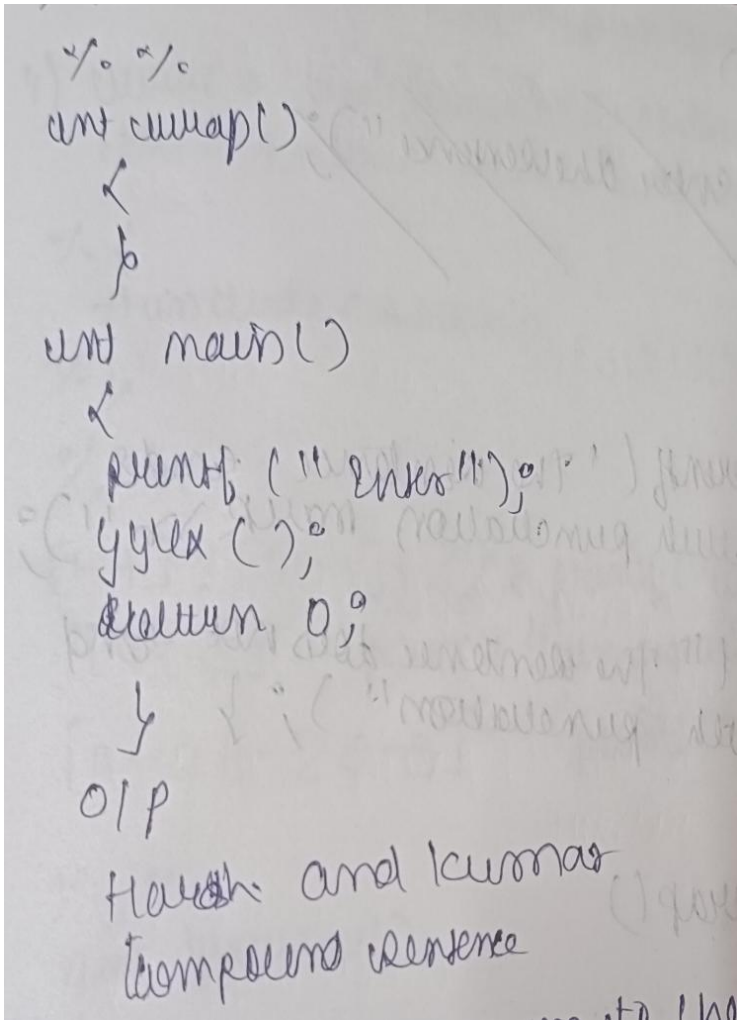
Output

```
Enter a number:
23
Not a floating point number!

0.5
Floating point number!

.8
Floating point number!

-.9
Floating point number!

+56
Not a floating point number!
```

3.2 Read and input sentence, and check if it is compound or simple. If a sentence has the word- and , or ,but ,because ,if ,then ,nevertheless then it is compound else it is simple.

Code



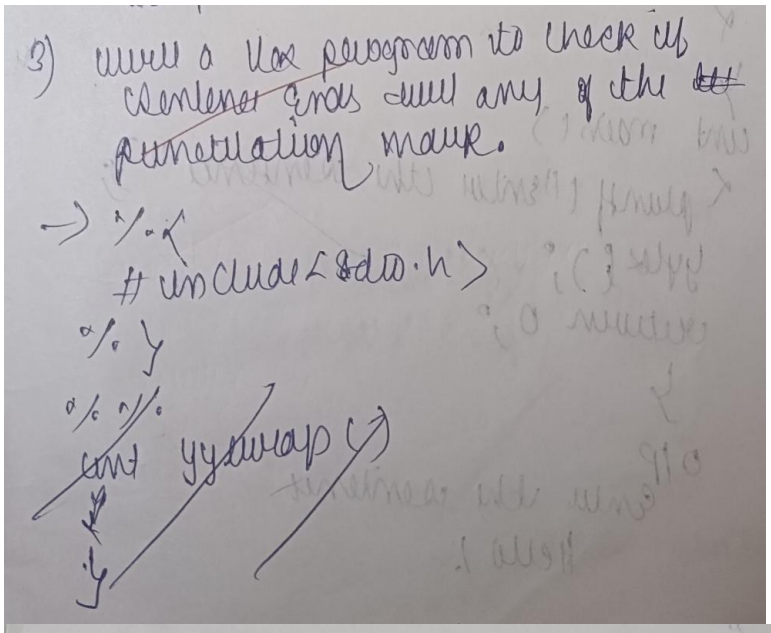


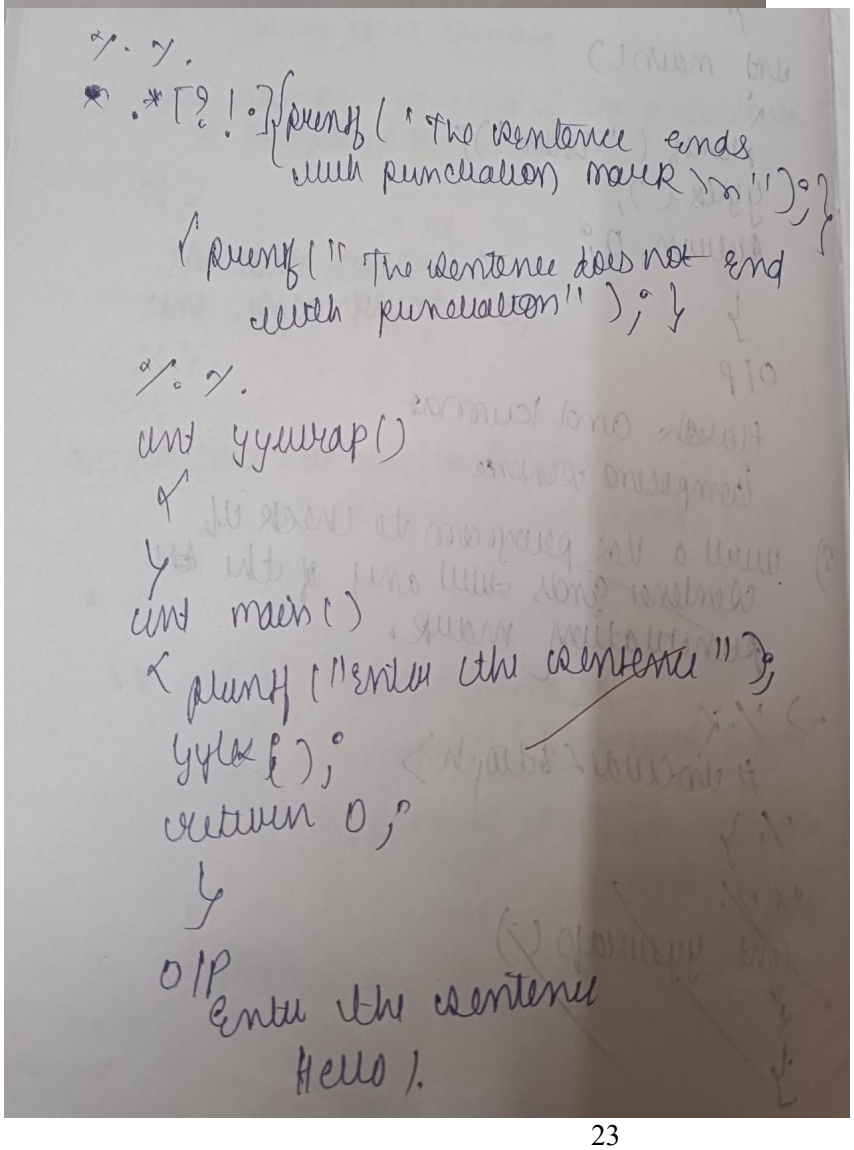
Output

```
Enter a sentence:
This is a car.
Simple sentence!
Enter a sentence:
She is good at singing and dancing.
Compound sentence!
```

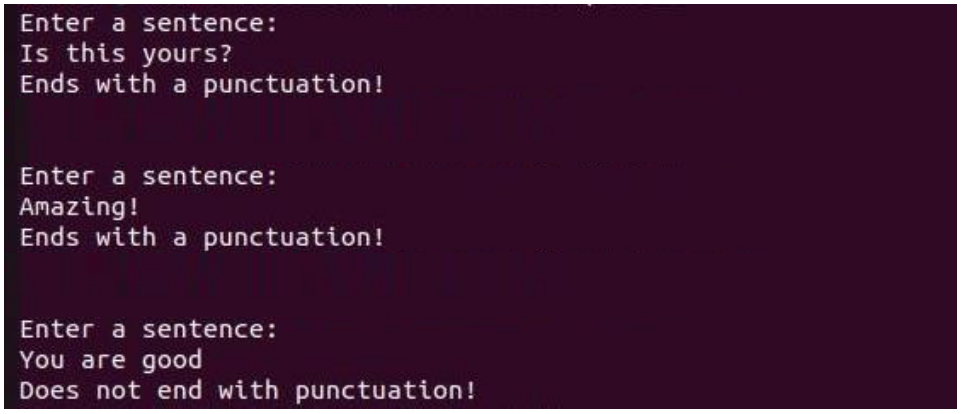
3.3 Write a program to check if the input sentence ends with any of the following punctuation marks (? , fullstop , !)

Code



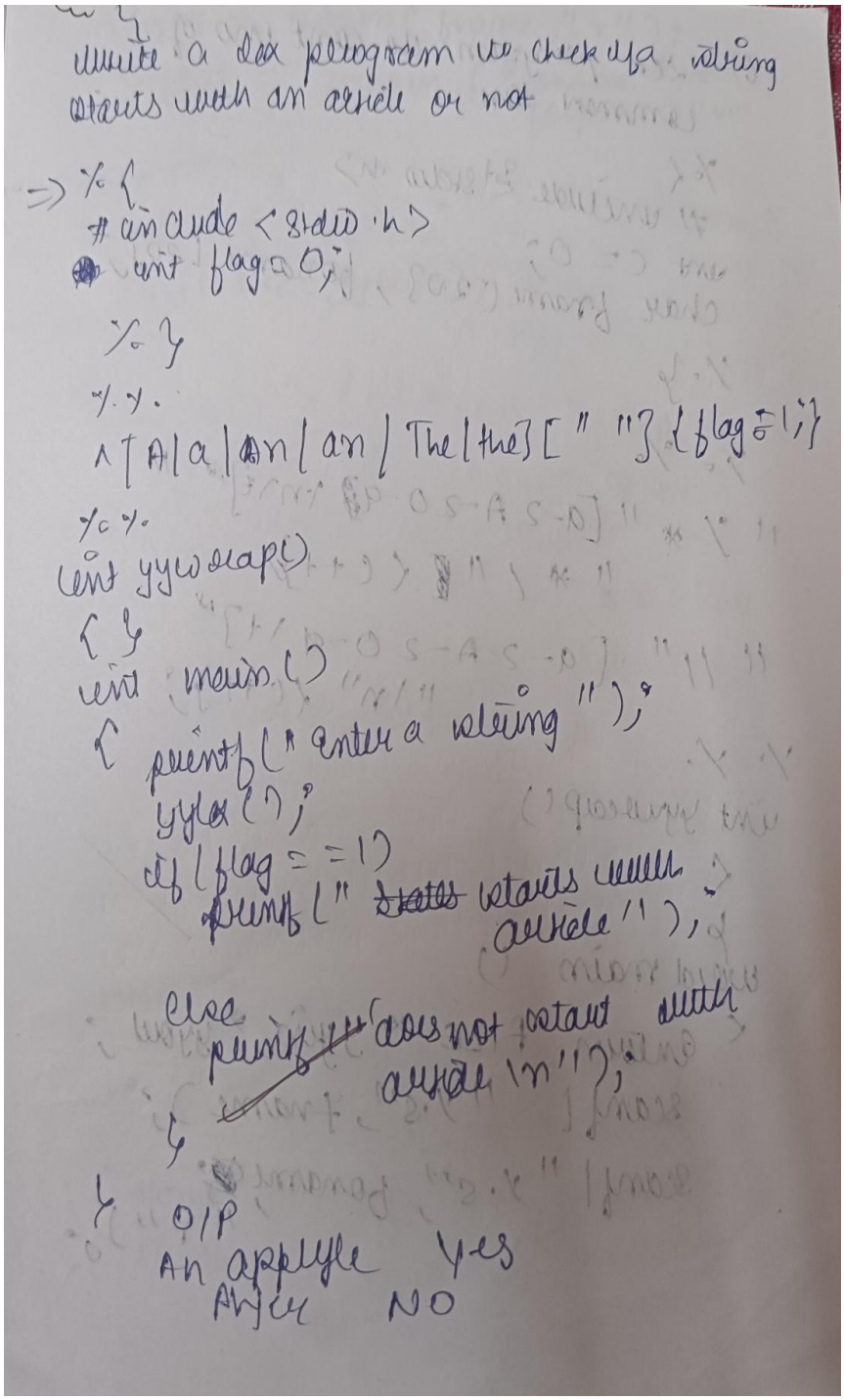


Output

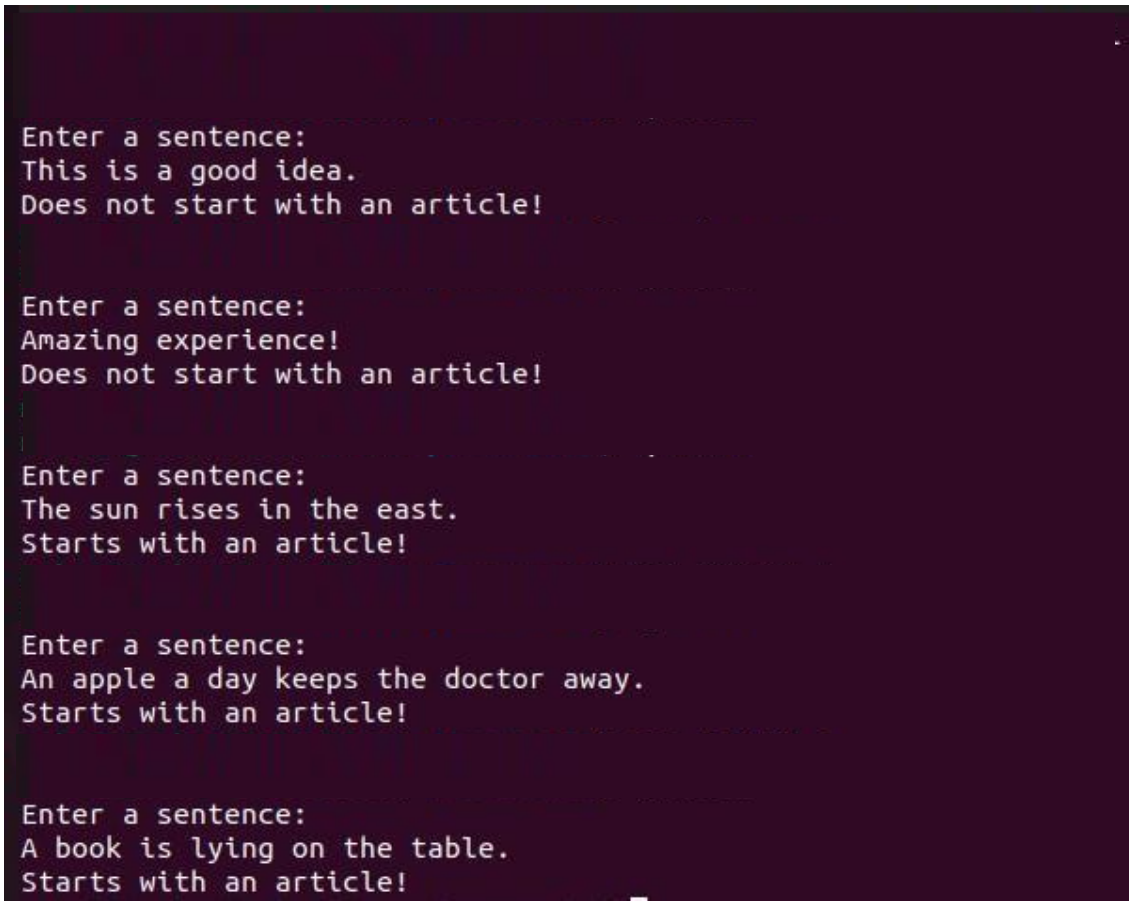

Enter a sentence:
Is this yours?
Ends with a punctuation!
Enter a sentence:
Amazing!
Ends with a punctuation!
Enter a sentence:
You are good
Does not end with punctuation!

3.4 Write a program to read an input sentence and to check if the sentence begins with English articles (A, a,AN,An,THE and The).

Code

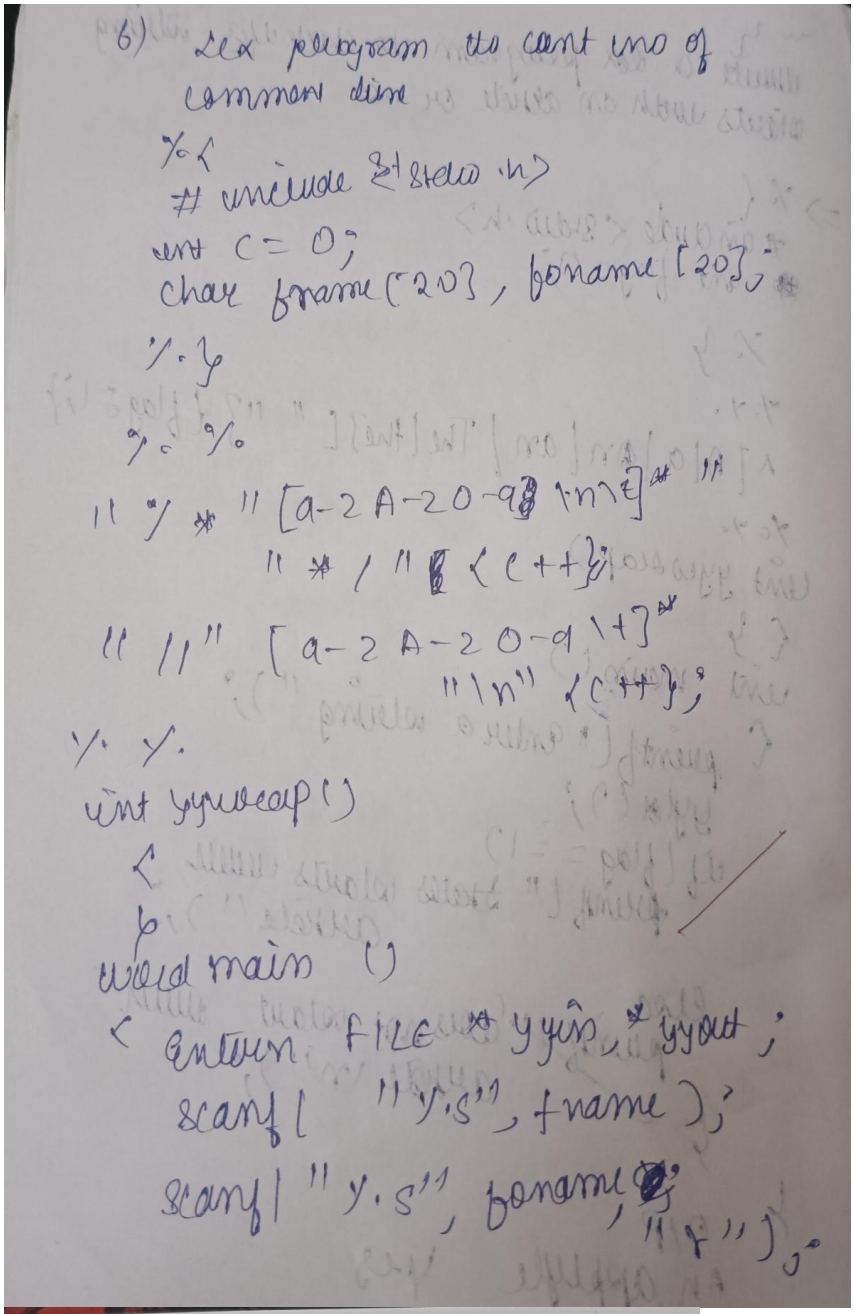


Output

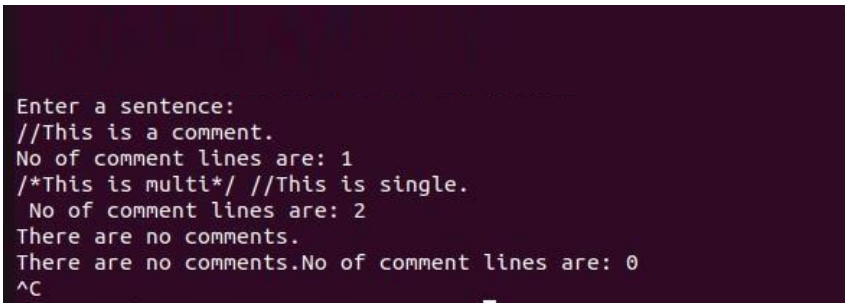

Enter a sentence:
This is a good idea.
Does not start with an article!
Enter a sentence:
Amazing experience!
Does not start with an article!
Enter a sentence:
The sun rises in the east.
Starts with an article!
Enter a sentence:
An apple a day keeps the doctor away.
Starts with an article!
Enter a sentence:
A book is lying on the table.
Starts with an article!

3.5 Lex program to count the number of comment lines (multi line comments or single line) in a program. Read the input from a file called input.txt and print the count in a file called output.txt.

Code

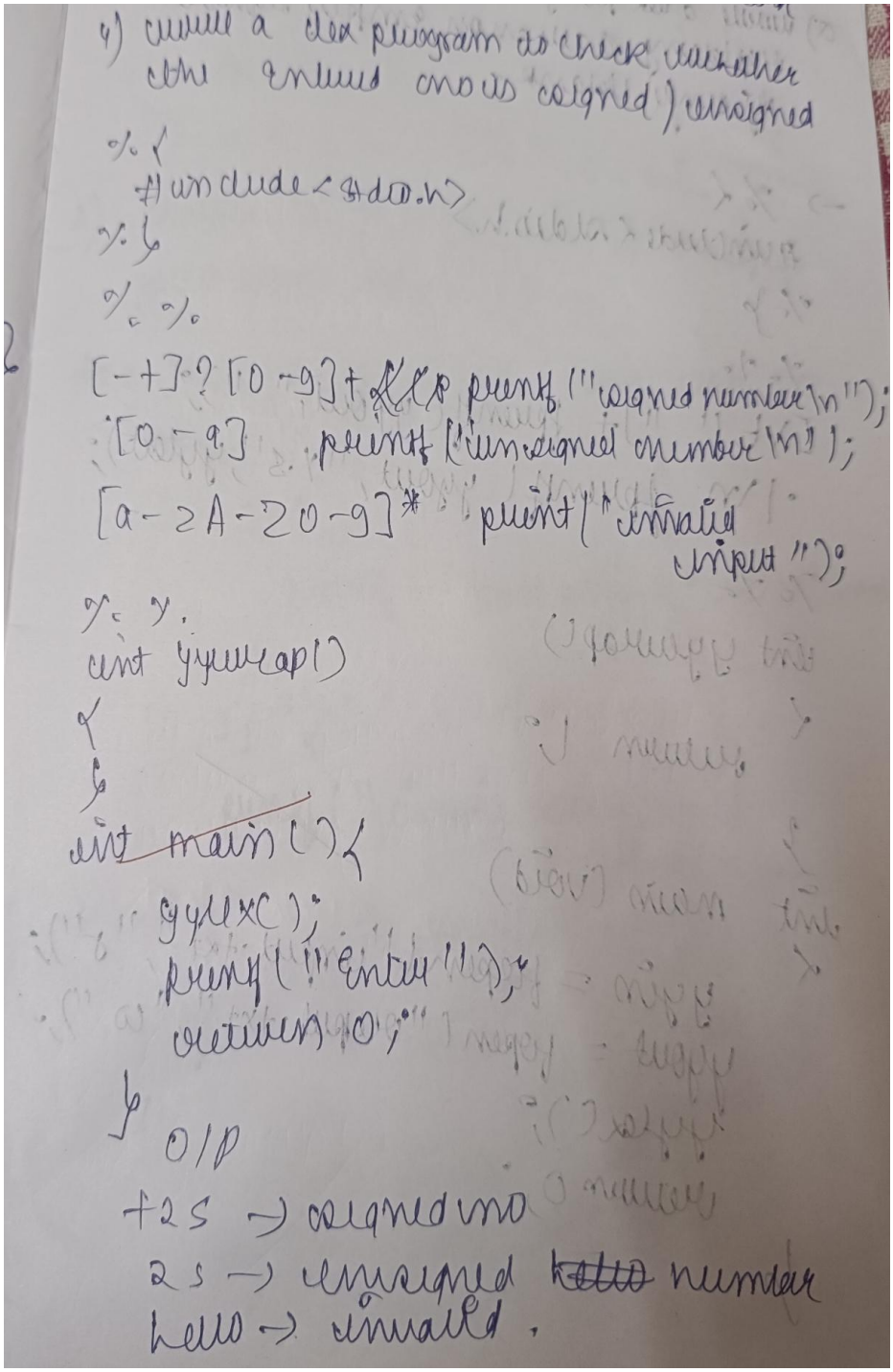


Output


Enter a sentence:
//This is a comment.
No of comment lines are: 1
/*This is multi*/ //This is single.
No of comment lines are: 2
There are no comments.
There are no comments.No of comment lines are: 0
^C

3.6 Write a program to read and check if the user entered number is signed or unsigned using appropriate meta character.

Code



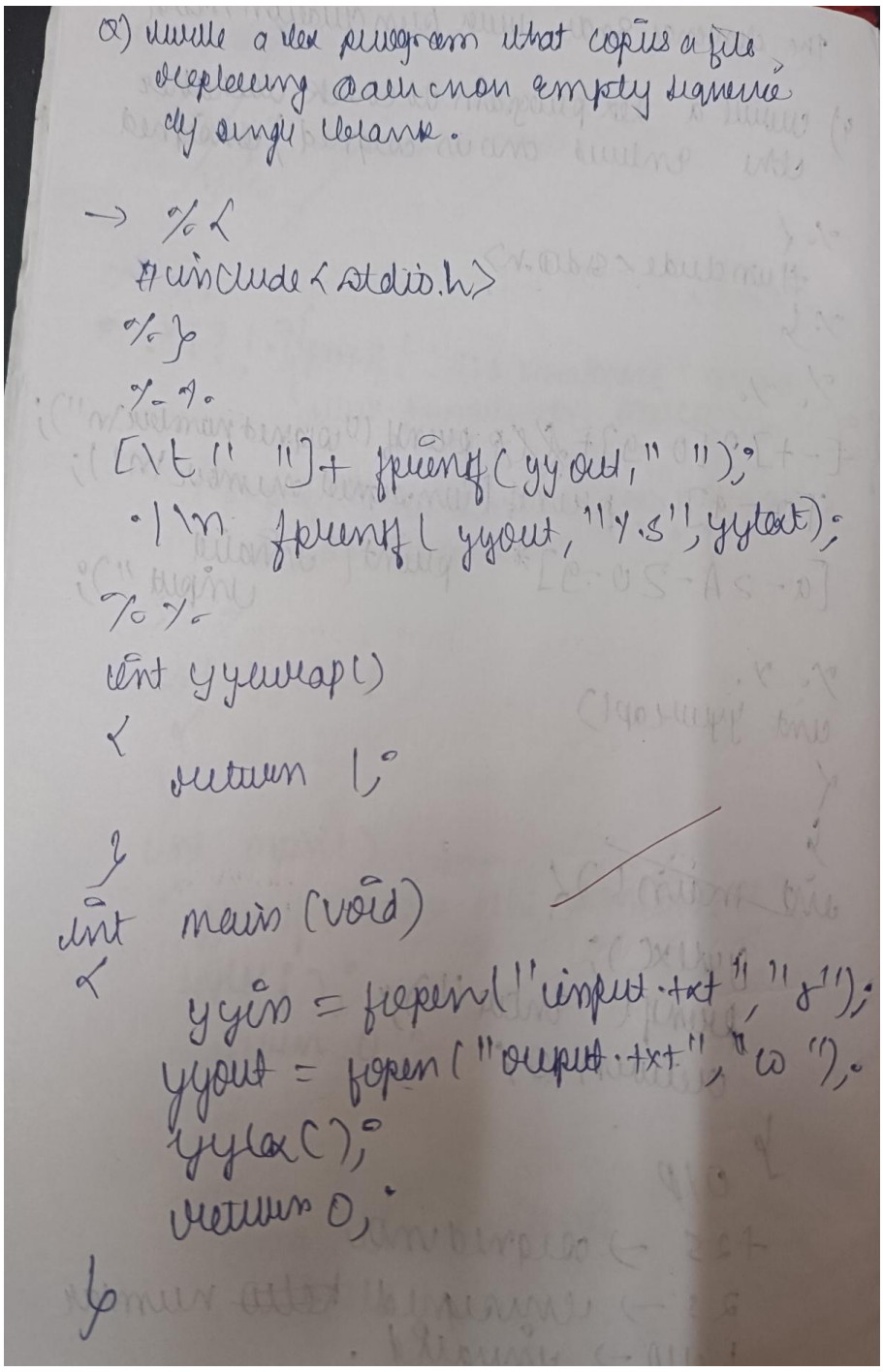
Output


Enter a number:
123
Unsigned number!
-123
Signed number!
+123
Signed number!
^C

Lab 4

4.1 Write a LEX program that copies a file, replacing each nonempty sequence of white spaces by a single blank.

Code



Output


*text.txt
1 Hello World
2 Welcome to programming

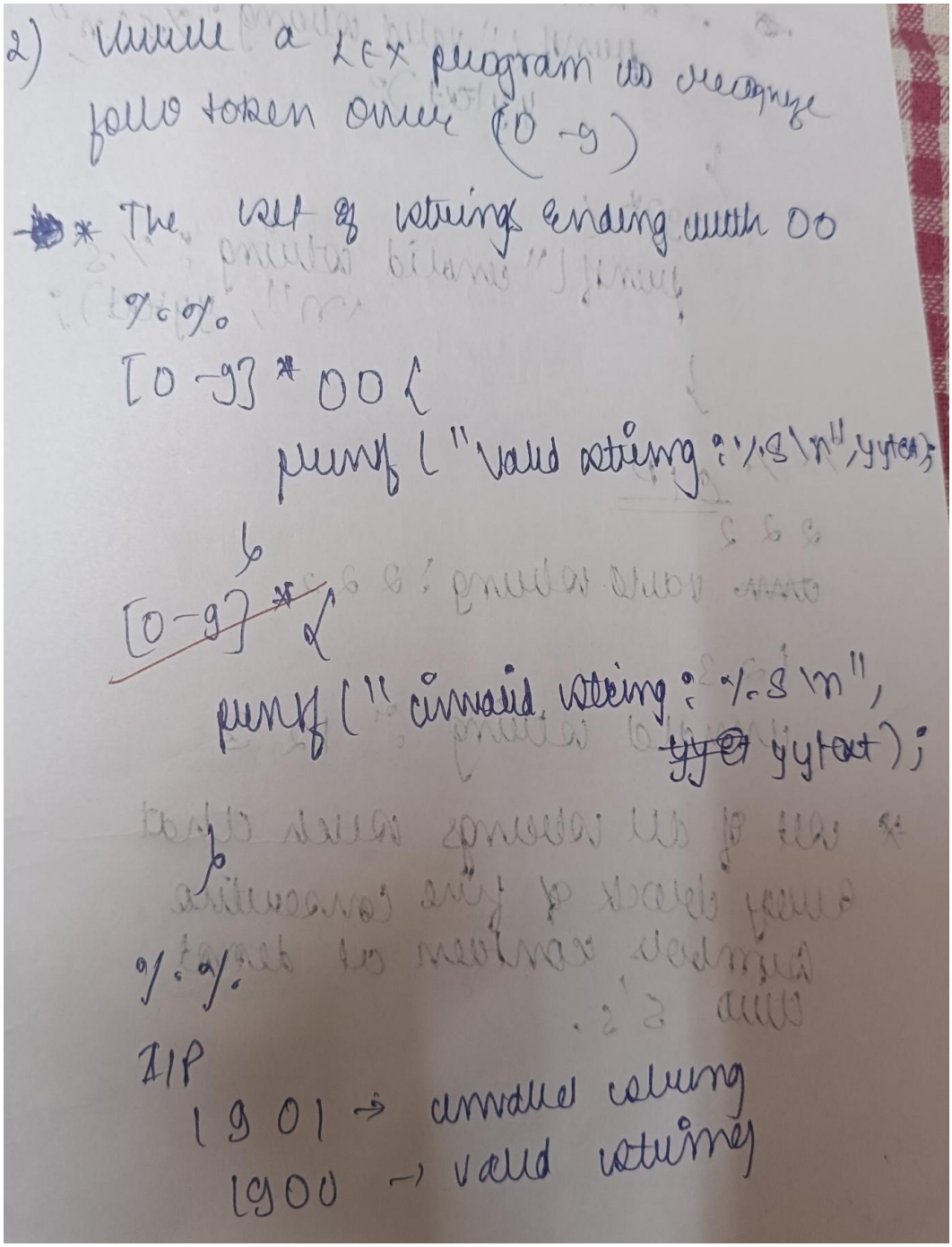

Printed!


Open
print.txt
~/Documents
1 Hello World
2 Welcome to programming

4.2 Write a LEX program to recognize the following tokens over the alphabets {0,1,..,9}

4.2.1 The set of all string ending in 00.

Code

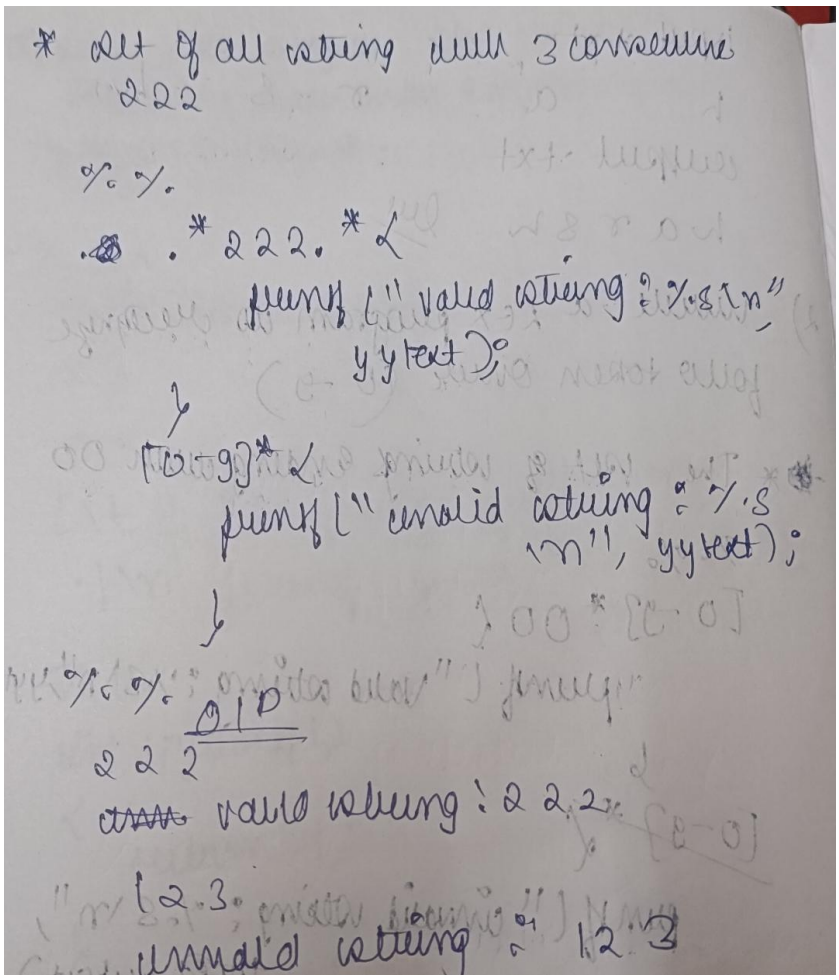


Output


Enter a string:
12300
Ends with 0.
Enter a string:
145
Does not end with 0.

4.2.2 The set of all strings with three consecutive 222's.

Code

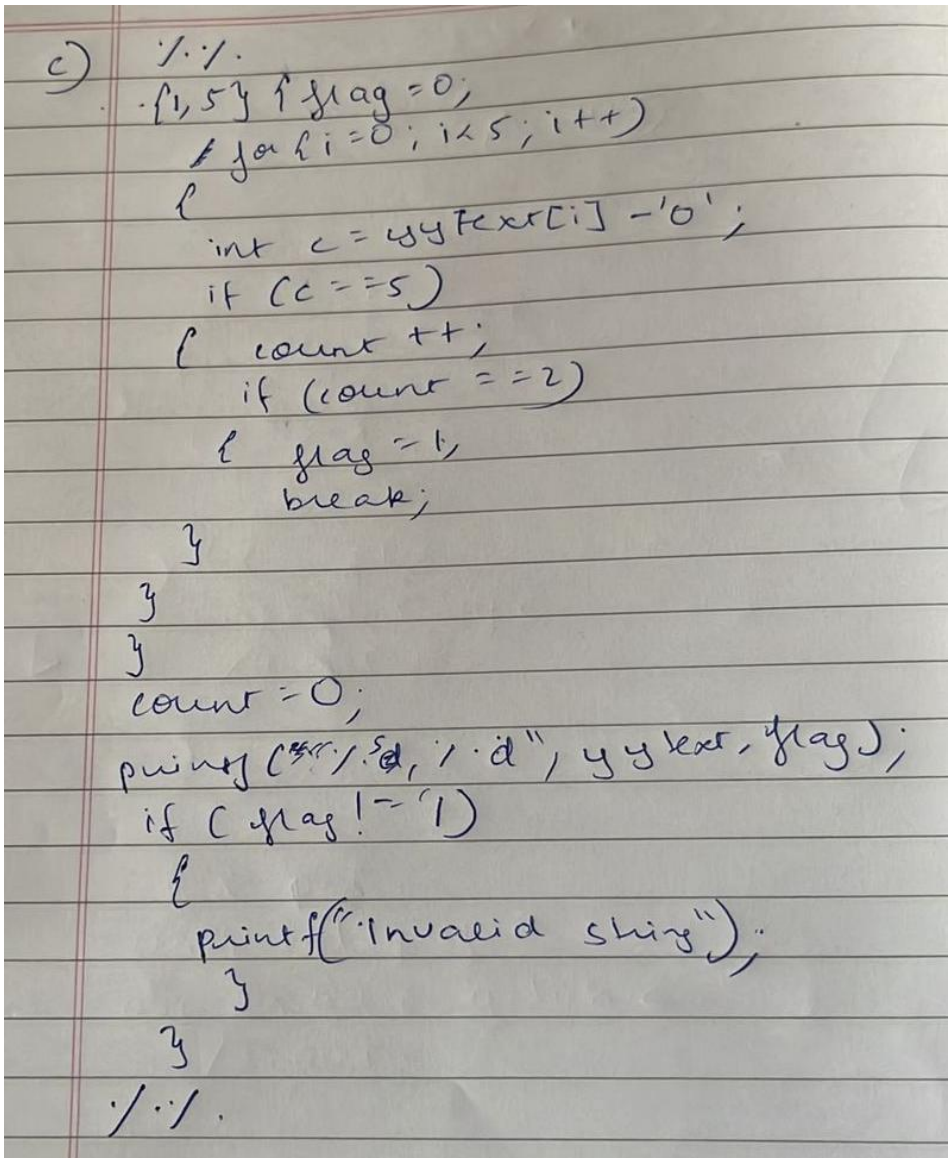


Output

```
Enter a string:
2322
Does not have 3 consecutive 2's.


Enter a string:
322221
Has 3 consecutive 2's.
```

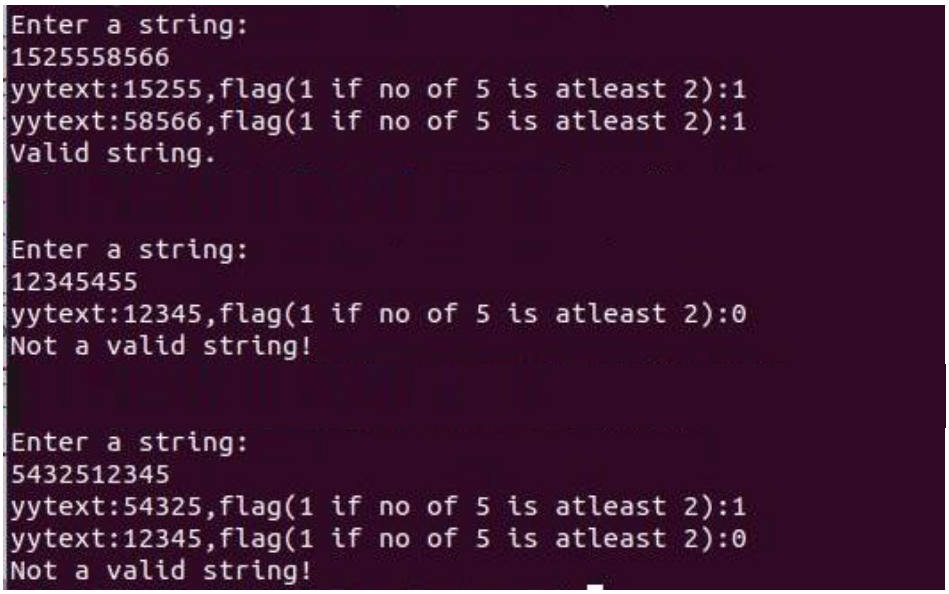
4.2.3 The set of all string such that every block of five consecutive symbols contains at least two 5's.

Code



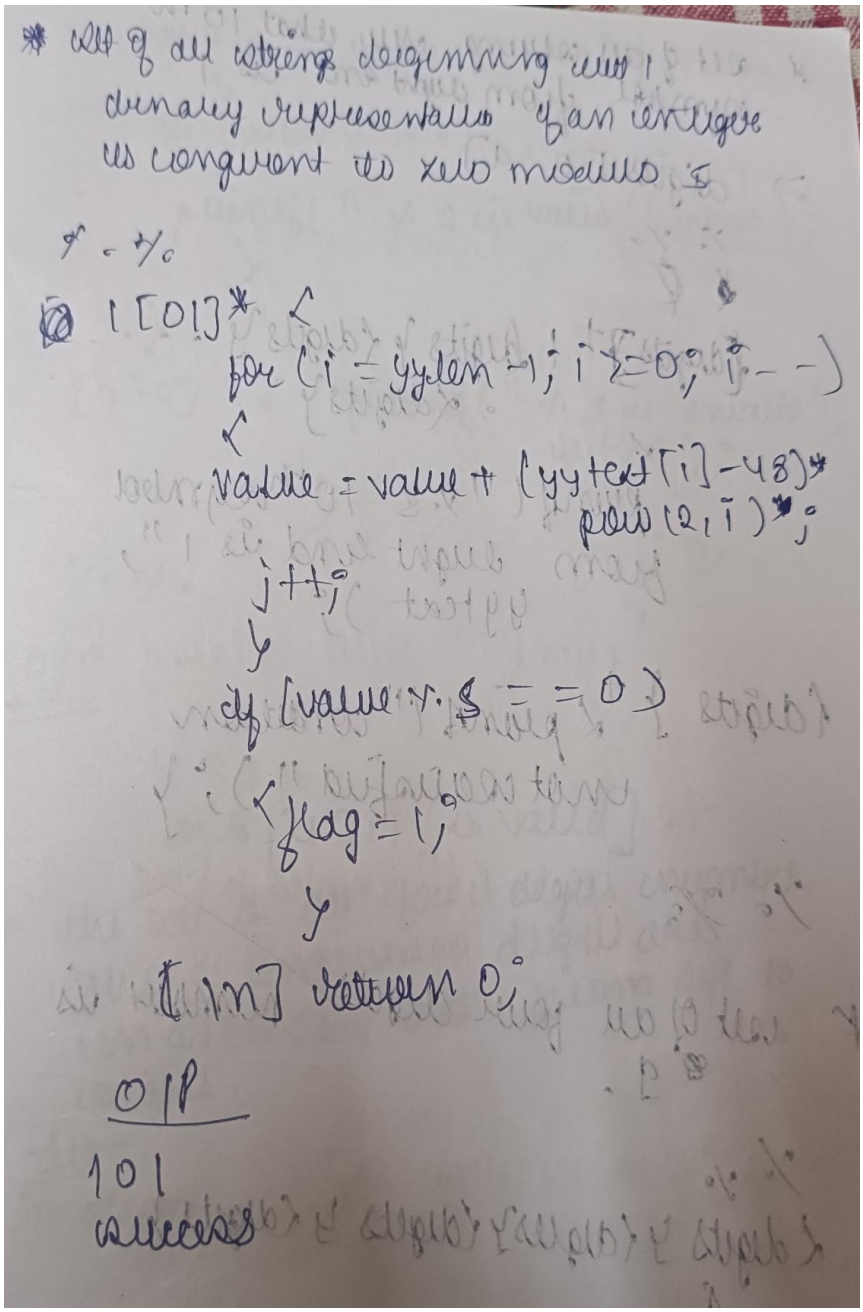
```
c) %%
[1,5] {flag=0;
  for(i=0; i<5; i++)
  {
   int c = yytext[i] - '0';
   if (c==5)
   { count++;
     if (count==2)
     { flag=1;
       break;
     }
   }
  }
  count=0;
printf("%d, %d", yytext, flag);
 if (flag!='1')
  {
   printf("Invalid string");
  }
 }
%%
```

Output

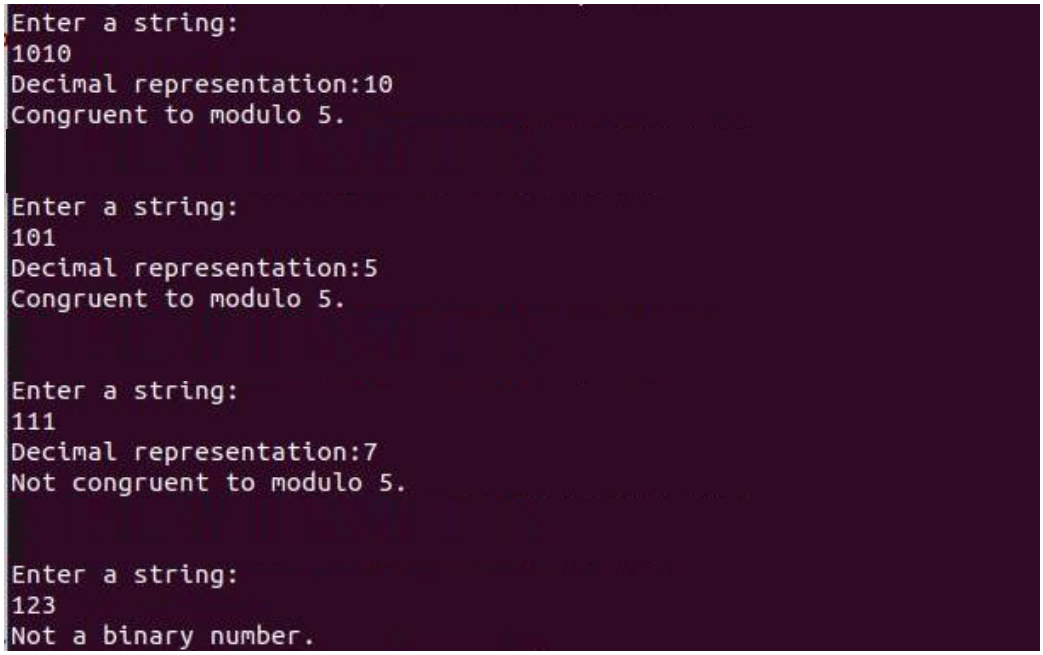

Enter a string:
1525558566
yytext:15255,flag(1 if no of 5 is atleast 2):1
yytext:58566,flag(1 if no of 5 is atleast 2):1
Valid string.
Enter a string:
12345455
yytext:12345,flag(1 if no of 5 is atleast 2):0
Not a valid string!
Enter a string:
5432512345
yytext:54325,flag(1 if no of 5 is atleast 2):1
yytext:12345,flag(1 if no of 5 is atleast 2):0
Not a valid string!

4.2.4 The set of all strings beginning with a 1 which, interpreted as the binary representation of an integer, is congruent to zero modulo 5.

Code

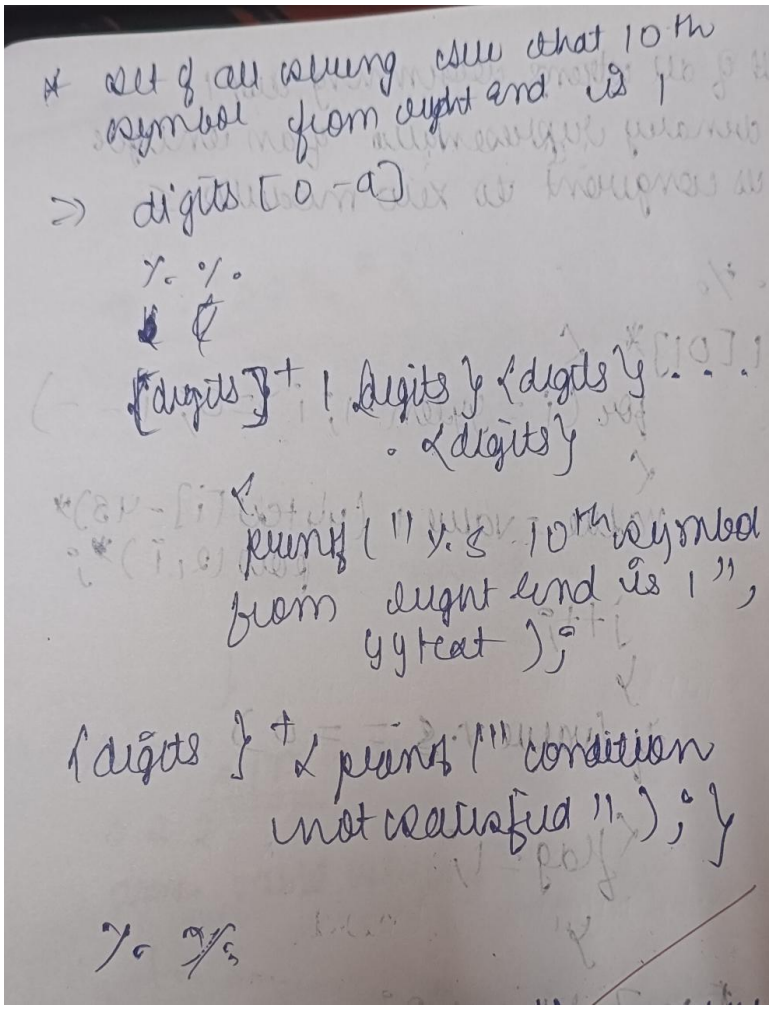


Output


Enter a string:
1010
Decimal representation:10
Congruent to modulo 5.
Enter a string:
101
Decimal representation:5
Congruent to modulo 5.
Enter a string:
111
Decimal representation:7
Not congruent to modulo 5.
Enter a string:
123
Not a binary number.

4.2.5 The set of all strings such that the 10th symbol from the right end is 1.

Code

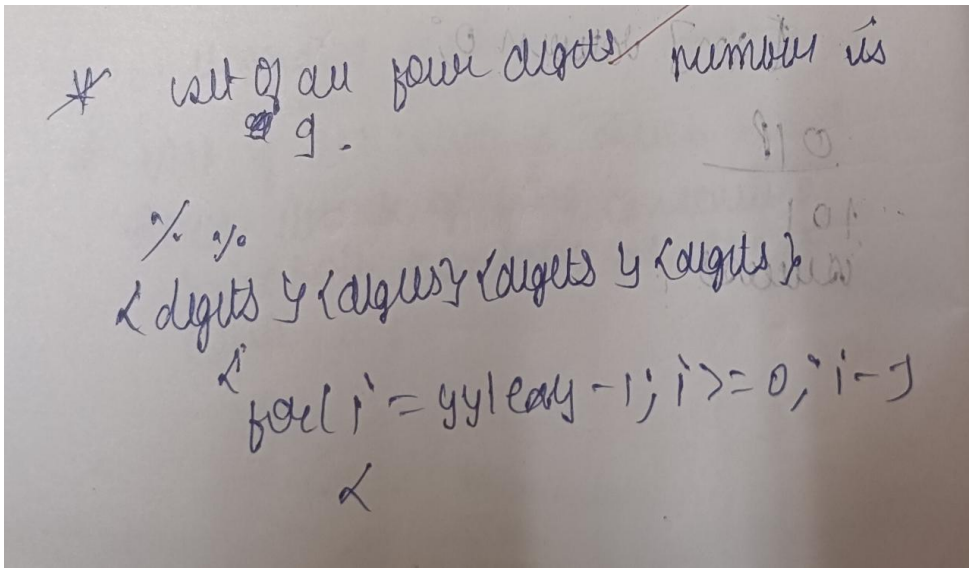


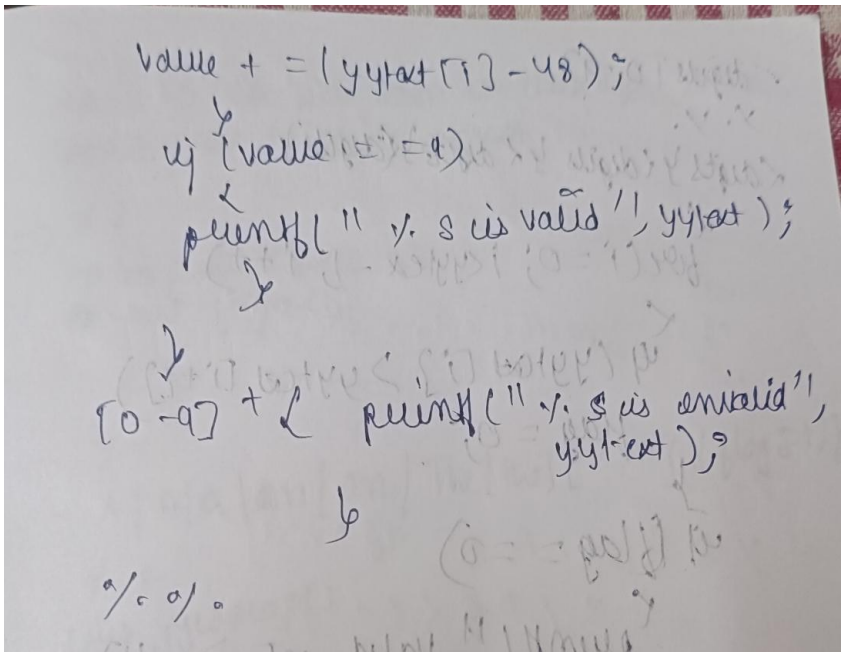
Output

```
Enter a string:
11234345236
10th symbol from right is 1.


Enter a string:
23123456123
10th symbol from right is not 1.
```

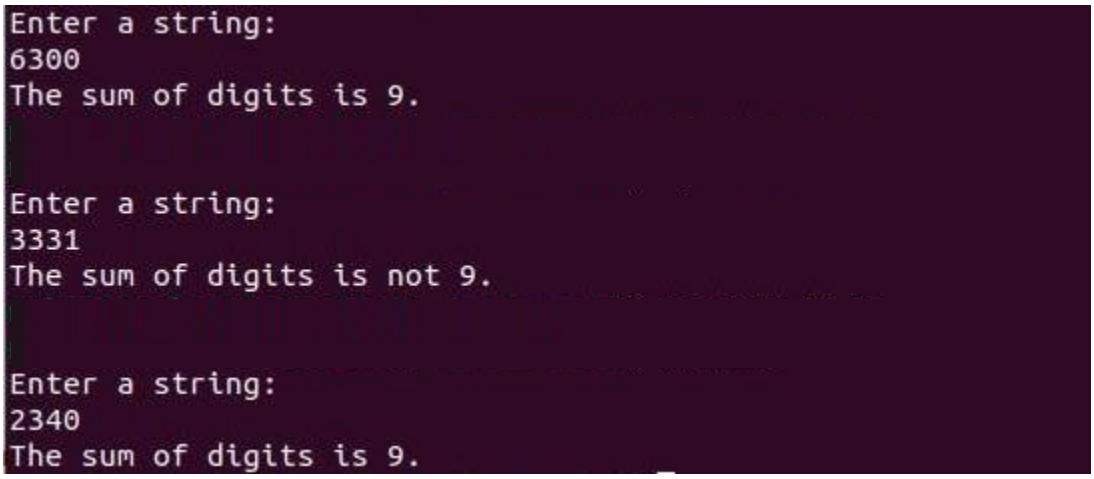
4.2.6 The set of all four digits numbers whose sum is 9.

Code



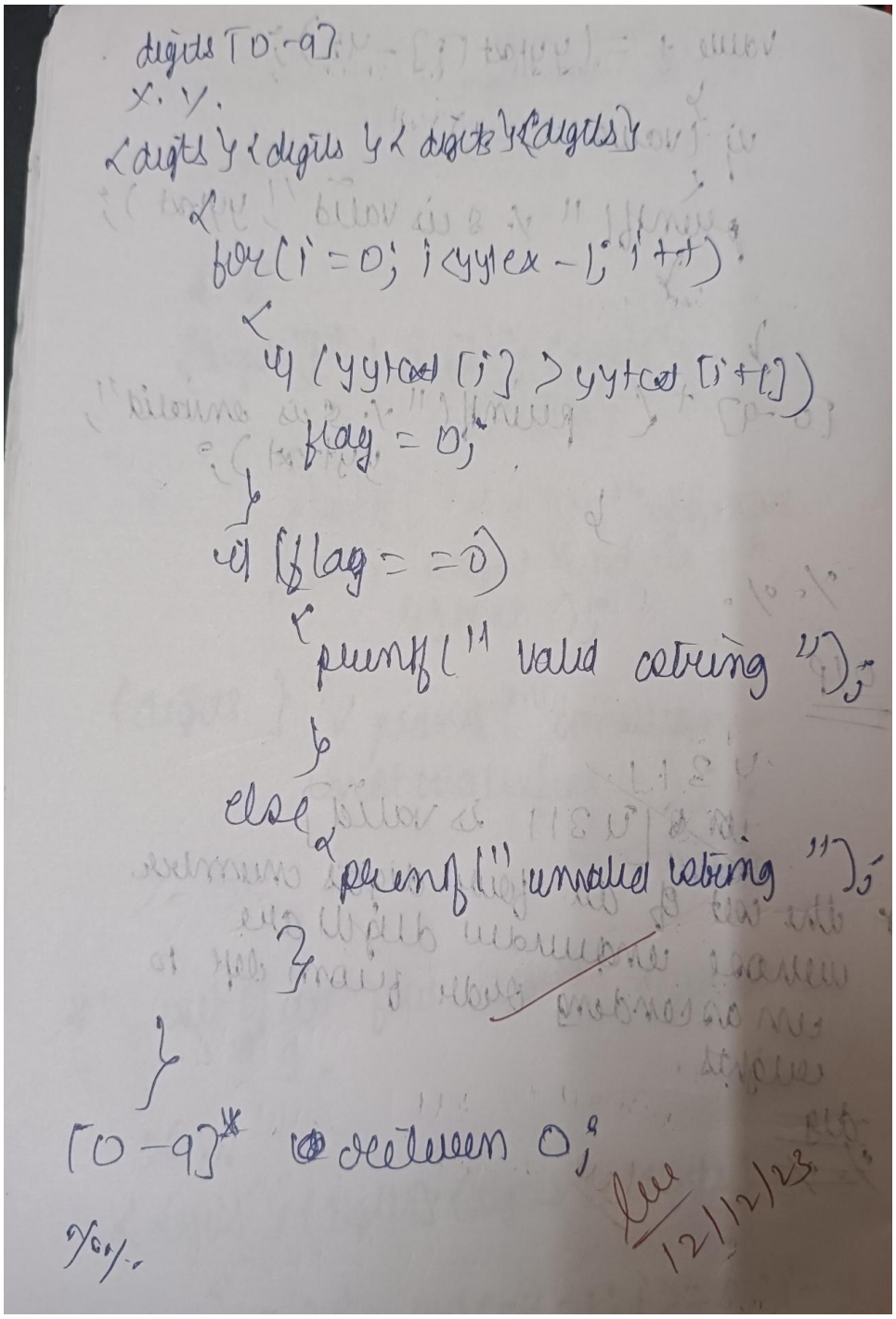


Output

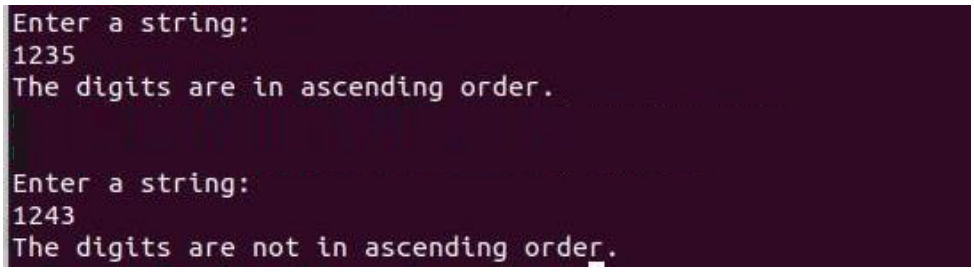

Enter a string:
6300
The sum of digits is 9.
Enter a string:
3331
The sum of digits is not 9.
Enter a string:
2340
The sum of digits is 9.

4.2.7 The set of all four digital numbers, whose individual digits are in ascending order from left to right.

Code



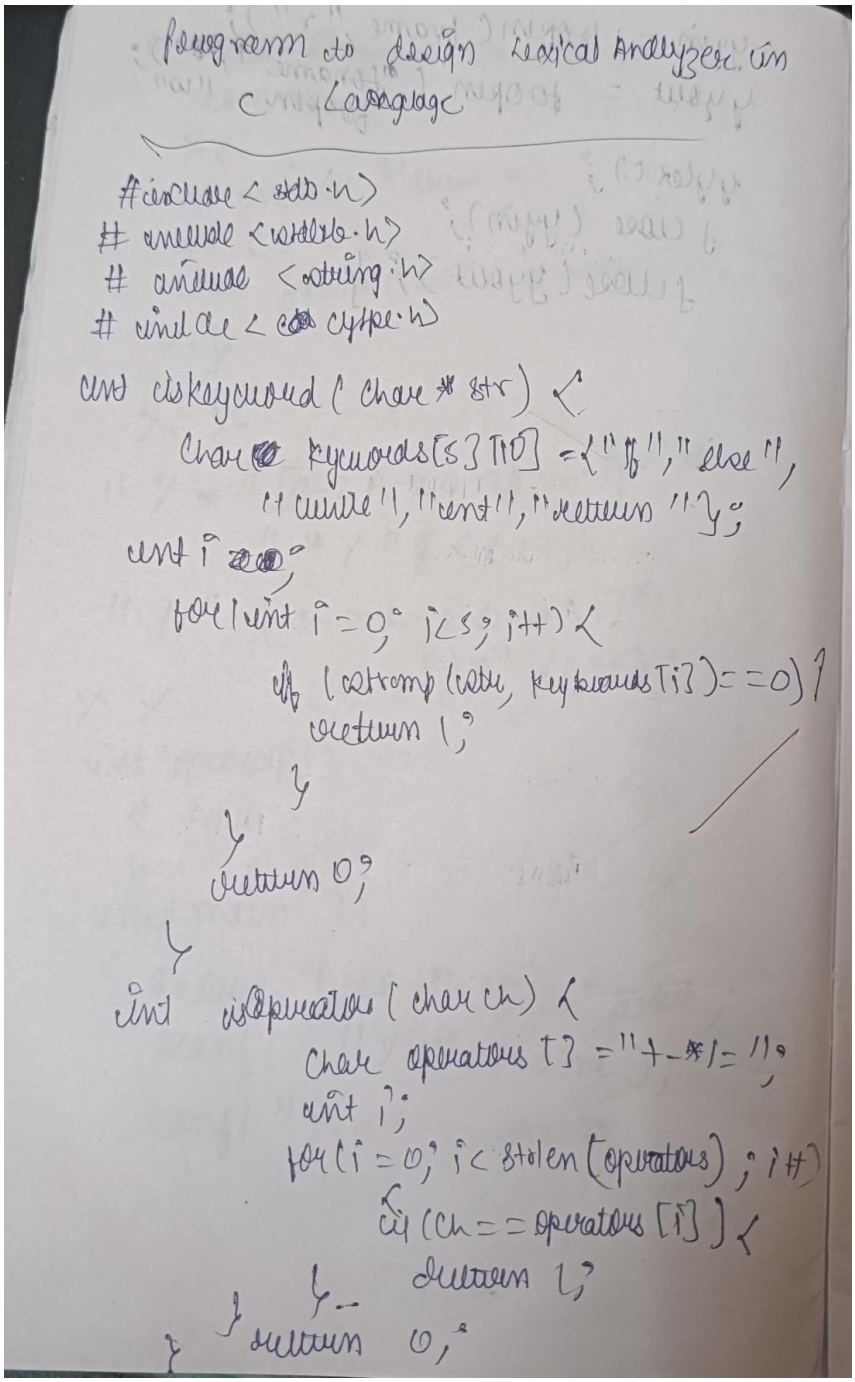
Output


Enter a string:
1235
The digits are in ascending order.
Enter a string:
1243
The digits are not in ascending order.

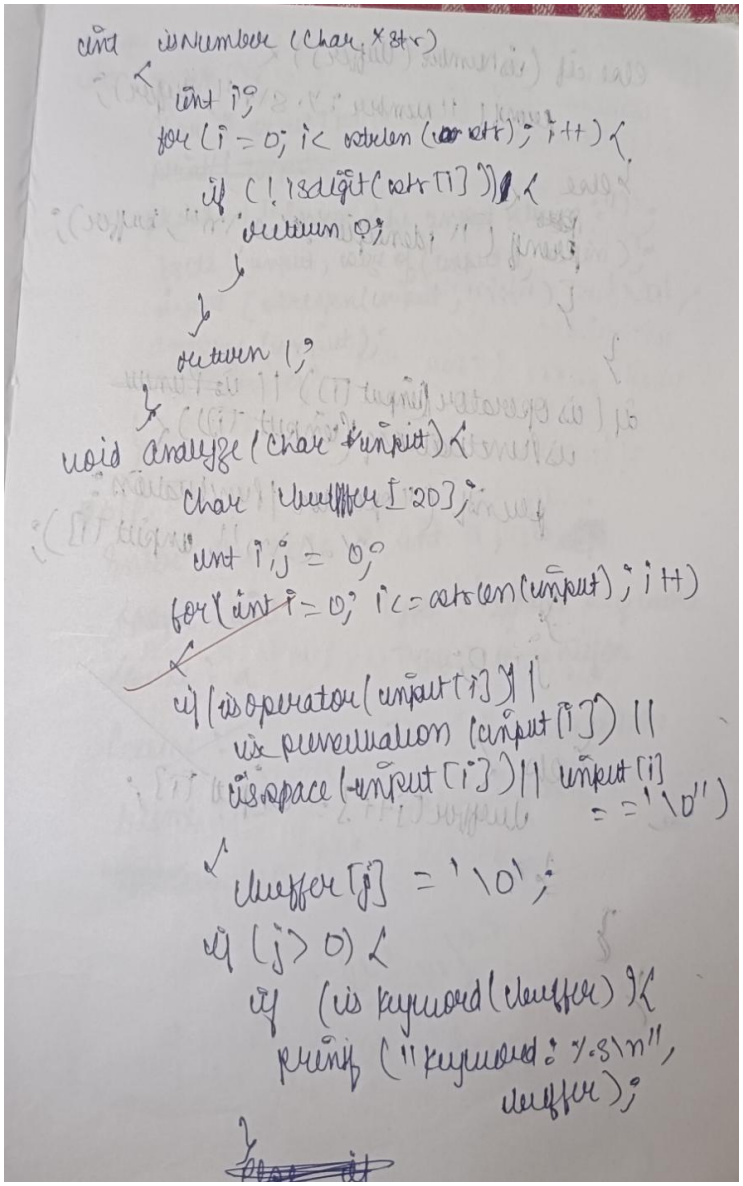
Lab 5

Write a C program to design lexical analysis to recognize any five keywords, identifiers, numbers, operators and punctuations.

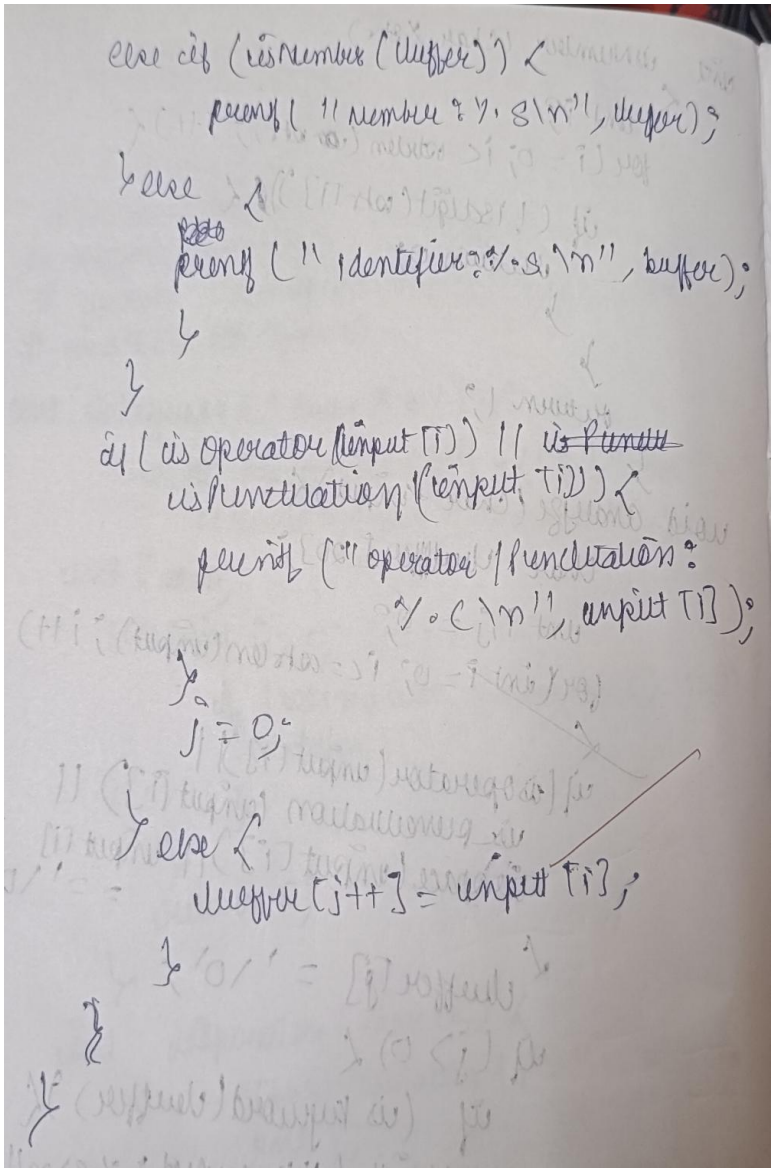
Code

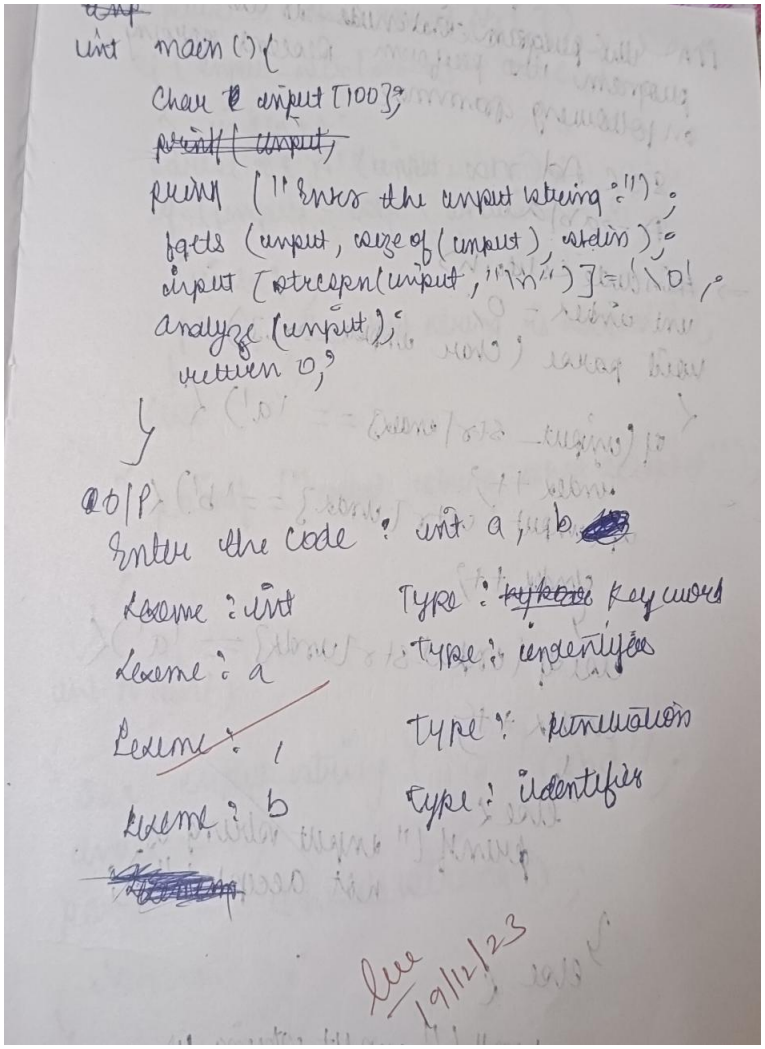


```
program to design Lexical Analyzer in
C Language

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

int iskeyword(char *str) {
    char keywords[5][10] = {"if", "else",
    "while", "int", "return"};
    int i;
    for(int i=0; i<5; i++) {
        if (strcmp(str, keywords[i])==0) {
            return 1;
        }
    }
    return 0;
}
int isOperator(char ch) {
    char operators[] = "+-*/=";
    int i;
    for(i=0; i< strlen(operators); i++)
        if(ch==operators[i]) {
            return 1;
        }
    return 0;
}
```



```
int isNumber(char *str)
{
    int i;
    for(i = 0; i < strlen(str); i++){
        if(!isdigit(str[i])){
            return 0;
        }
    }
    return 1;
}
void analyze(char *input){
    char buffer[20];
    int i, j = 0;
    for(int i = 0; i <= strlen(input); i++)
    {
        if(isoperator(input[i]) ||
           is punctuation(input[i]) ||
           isspace(input[i]) || input[i]
                                == '\0')
        {
            buffer[j] = '\0';
            if(j > 0){
                if(is keyword(buffer)){
                    printf("keyword: %s\n",
                           buffer);
```


else if (isNumber(buffer)) {
printf("number : %s\n", buffer);
} else {
printf("identifier : %s\n", buffer);
}
if (isOperator(input[i]) || isPunctuation(input[i])) {
printf("operator/punctuation : %c\n", input[i]);
}
j = 0;
} else {
buffer[j++] = input[i];
}
}



Output

```
Keyword: if
Operator: (
Identifier: x
Operator: >
Number: 0
Operator: )
Operator: {
Keyword: return
Identifier: x
Punctuation:;
Operator: }
Keyword: else
Operator: {
Keyword: return
Operator: -x
Punctuation:;
Operator: }
```

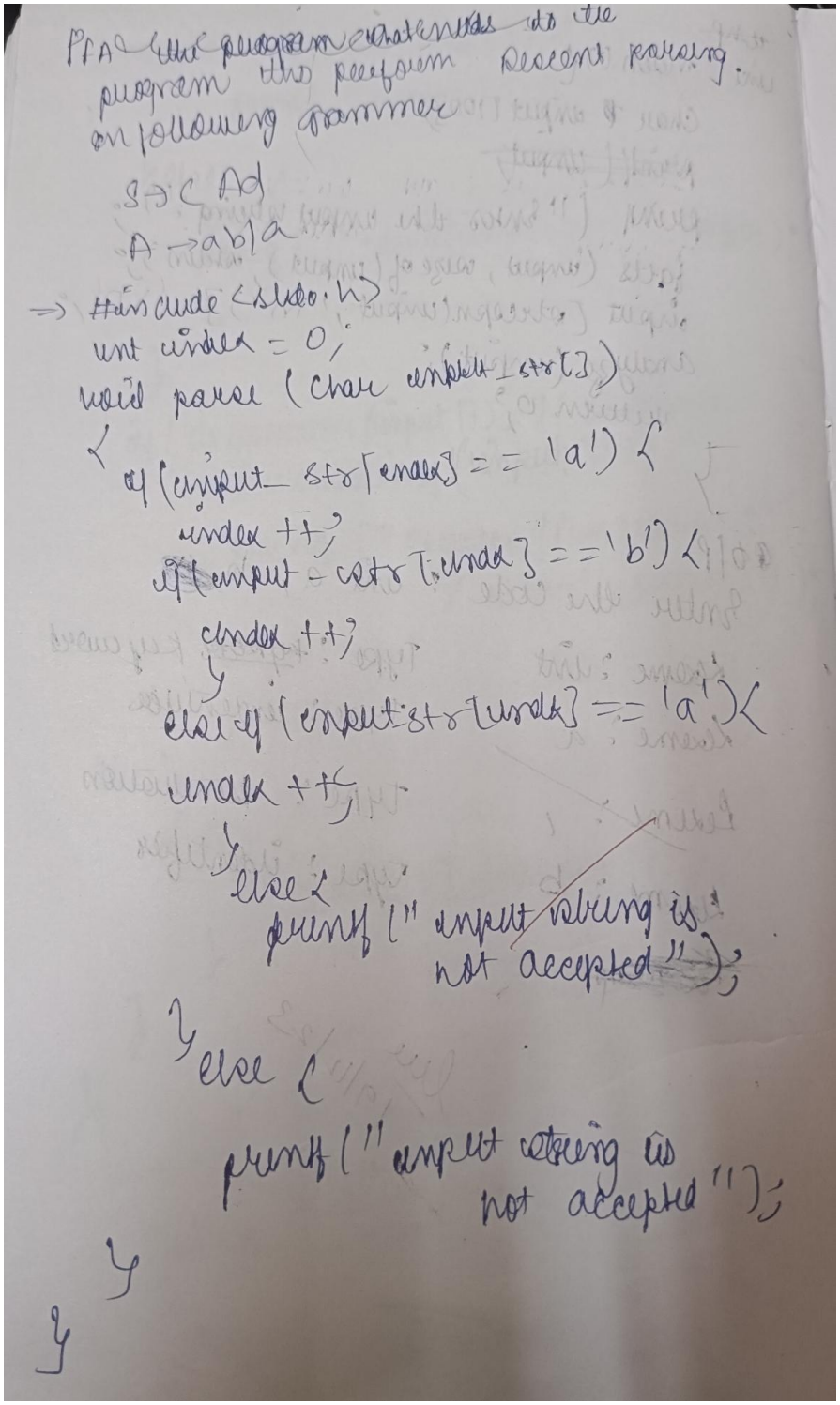
Lab 6

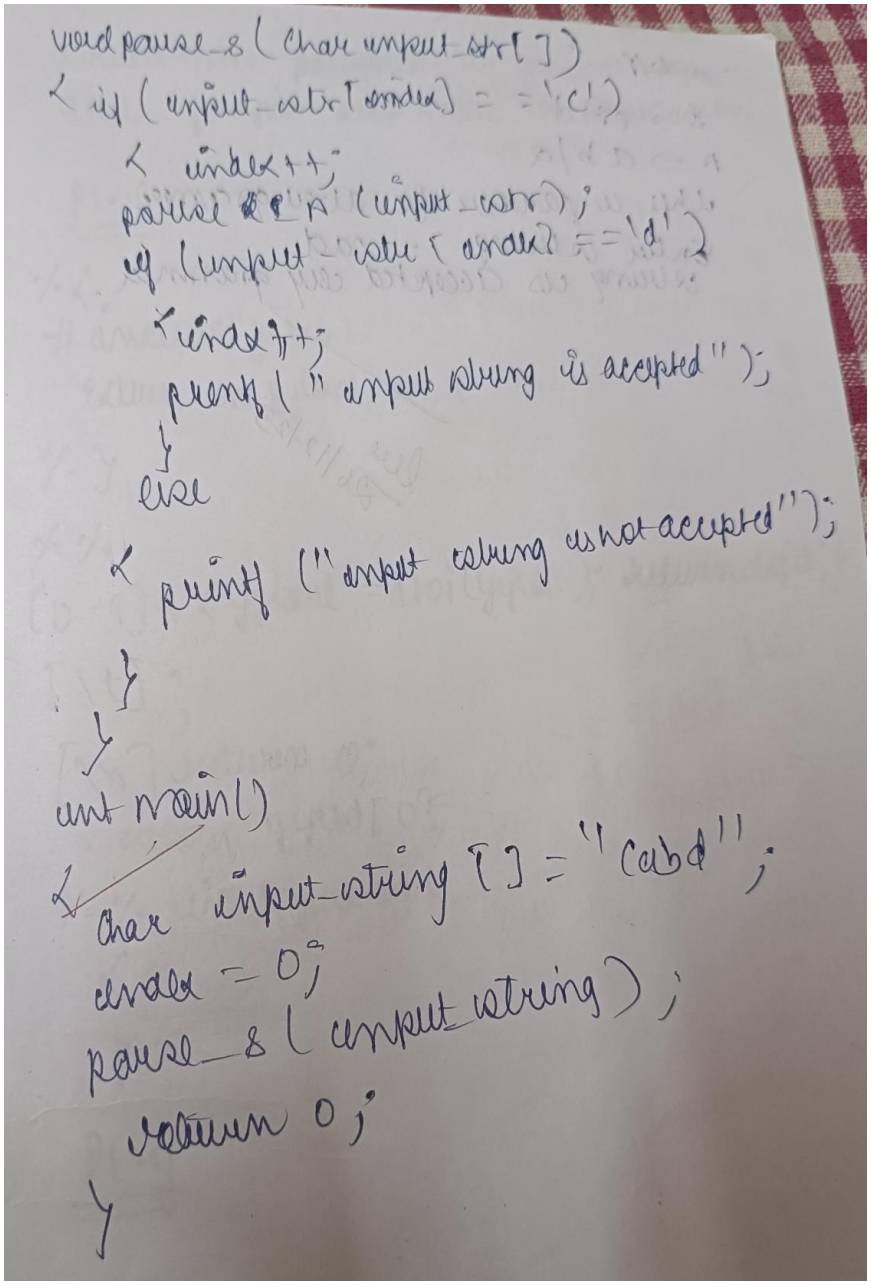
Write a program to perform recursive descent parsing on the following grammar:

S->cAd

A->ab | a

Code





```
void parse_s ( char input_str[ ])
{ if ( input_str[index] == 'c')
  { index++;
  parse_A (input_str);
  if (input_str[index] == 'd')
  { index++;
  printf ( " input string is accepted " );
  }
  else
  { printf (" input string is not accepted ");
  }
  }
int main()
{ char input_string [ ] = " cabd " ;
  index = 0;
  parse_s ( input_string ) ;
  return 0;
}
```

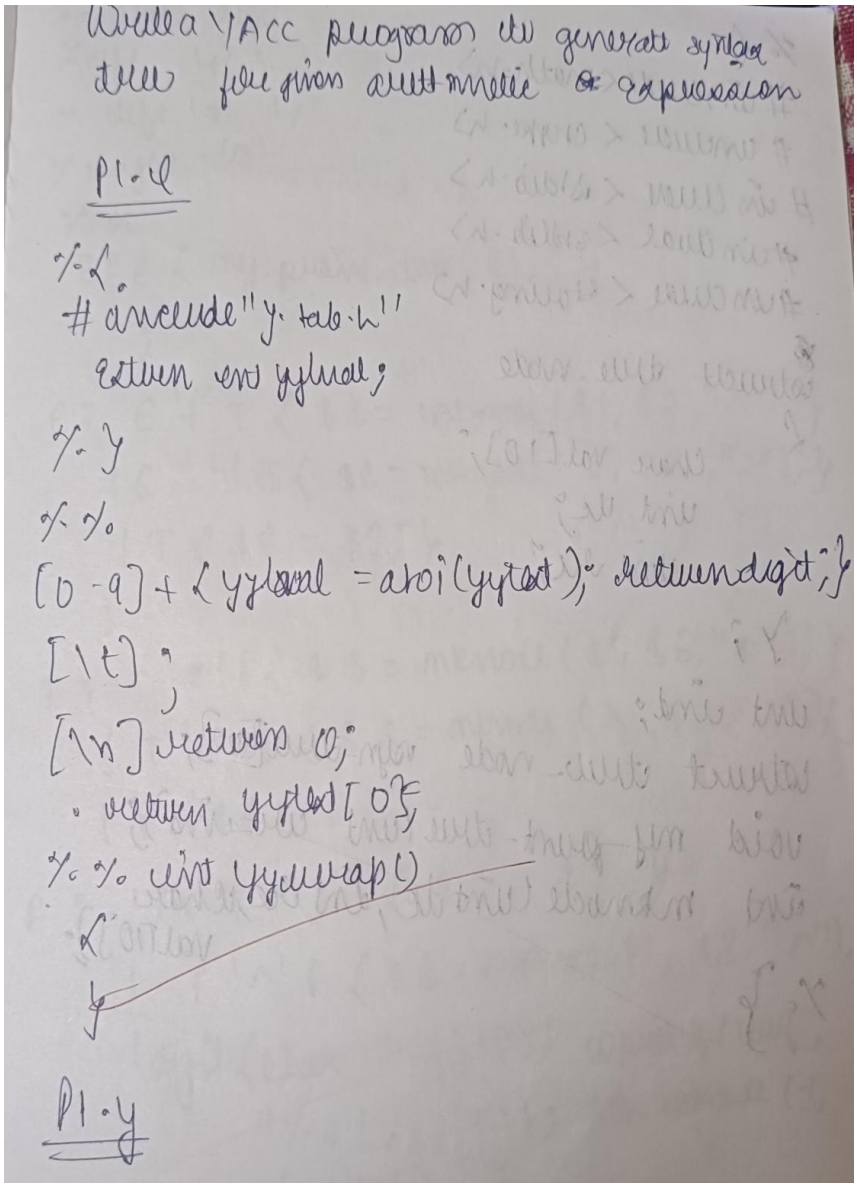
Output

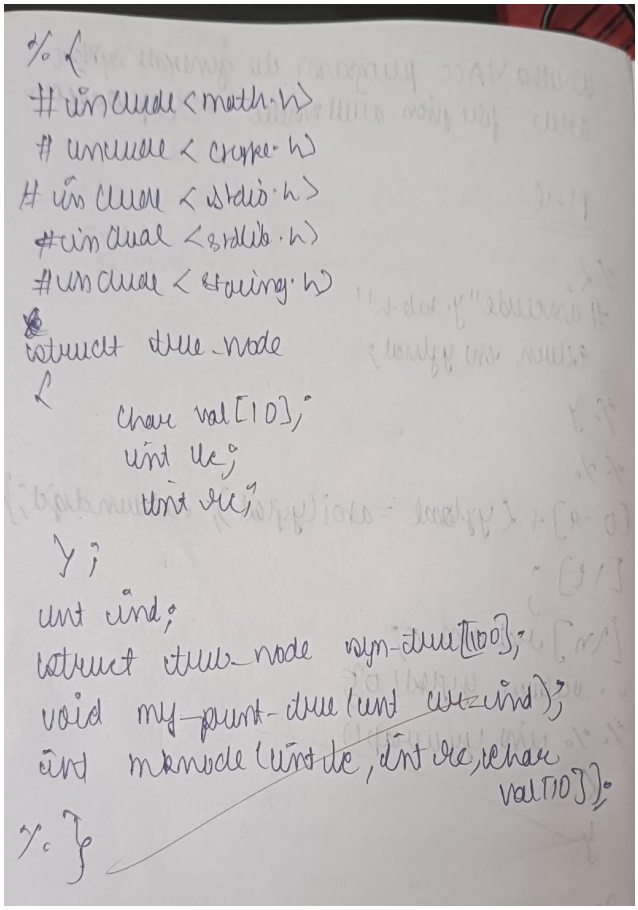
```
1.S->cAd
2.A->ab/a
Enter any string:cabd
String is accepted by the grammar
The productions used are
S -> cAd
A -> ab


...Program finished with exit code 1
Press ENTER to exit console.
```

Lab 7

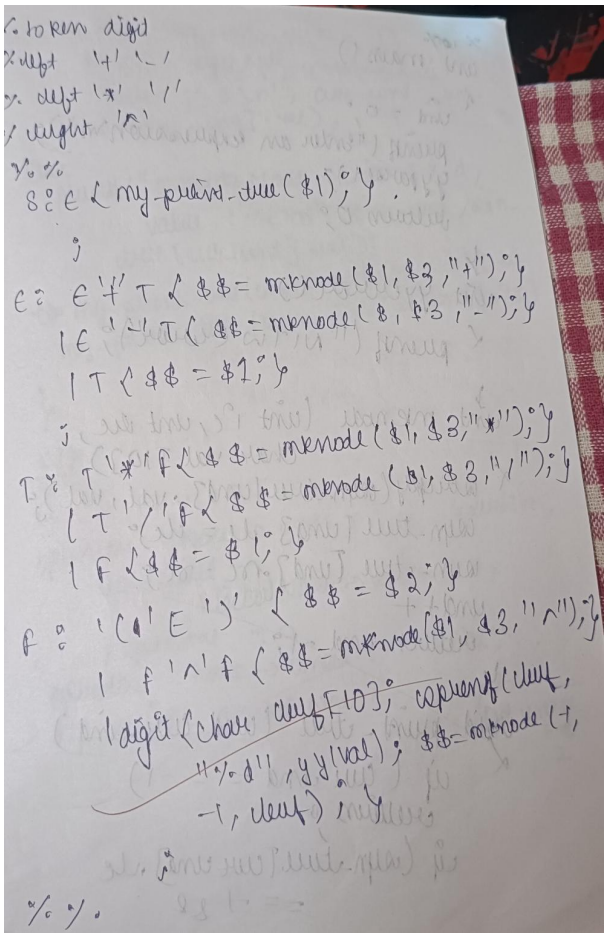
7.1 Write a program in YACC to design a suitable grammar for evaluation of arithmetic expression having +, -, * and /.

Code



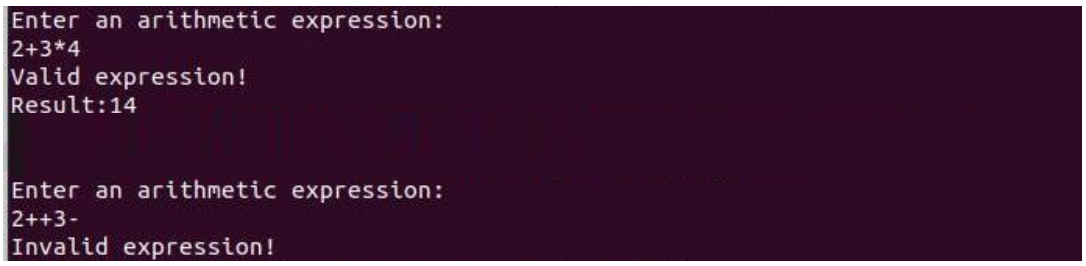


```
%{
#include<math.h>
#include<ctype.h>
#include<stdio.h>
#include<stdlib.h>
#include<string.h>

struct tree_node
{
    char val[10];
    int le;
    int ri;
};
int ind;
struct tree_node syn_tree[100];
void my_print_tree(int cur_ind);
int mknode(int le, int ri, char val[10]);
%}
```



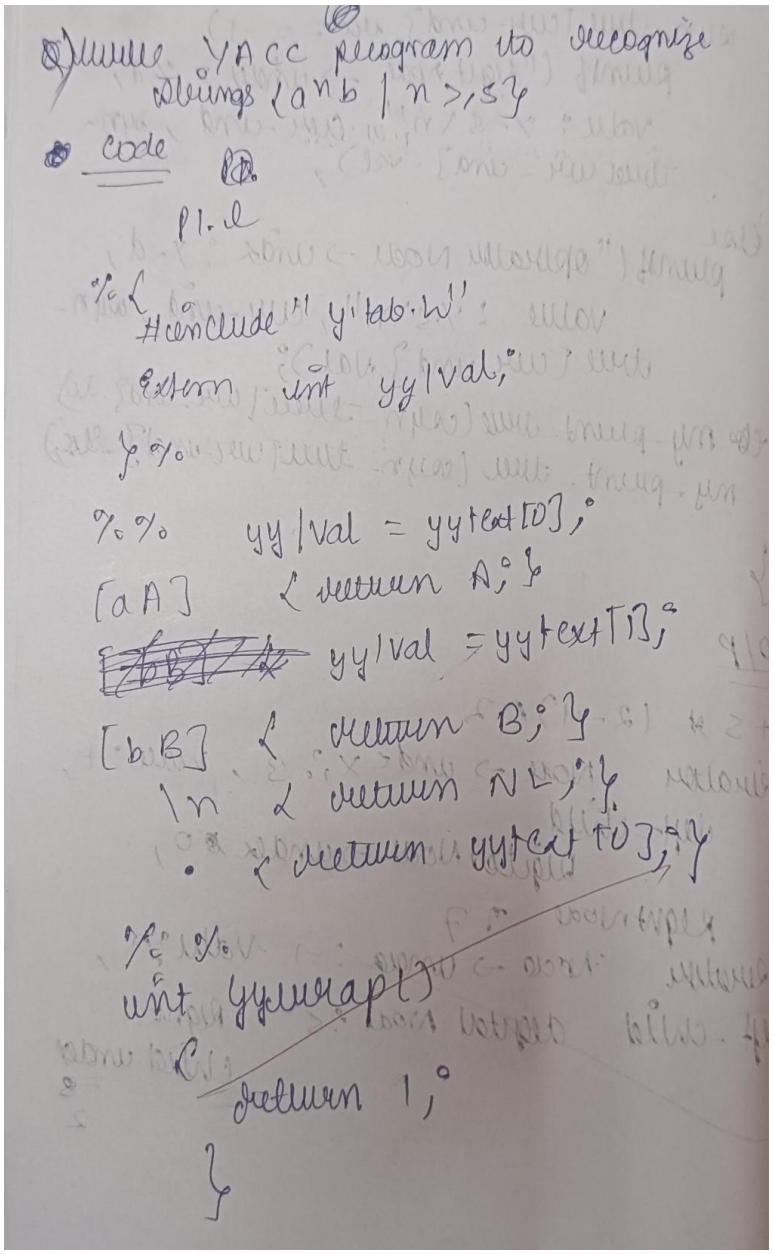
```
%token digit
%left '+' '-'
%left '*' '/'
%right '^'
%%
S: E { my_print_tree($1); }
 ;
E: E '+' T { $$ = mknode($1, $3, "+"); }
 | E '-' T { $$ = mknode($1, $3, "-"); }
 | T { $$ = $1; }
 ;
T: T '*' F { $$ = mknode($1, $3, "*"); }
 | T '/' F { $$ = mknode($1, $3, "/"); }
 | F { $$ = $1; }
F: '(' E ')' { $$ = $2; }
 | F '^' F { $$ = mknode($1, $3, "^"); }
 | digit { char buff[10]; sprintf(buff, "%d", yylval); $$ = mknode(-1, -1, buff); }
 ;
%%
```

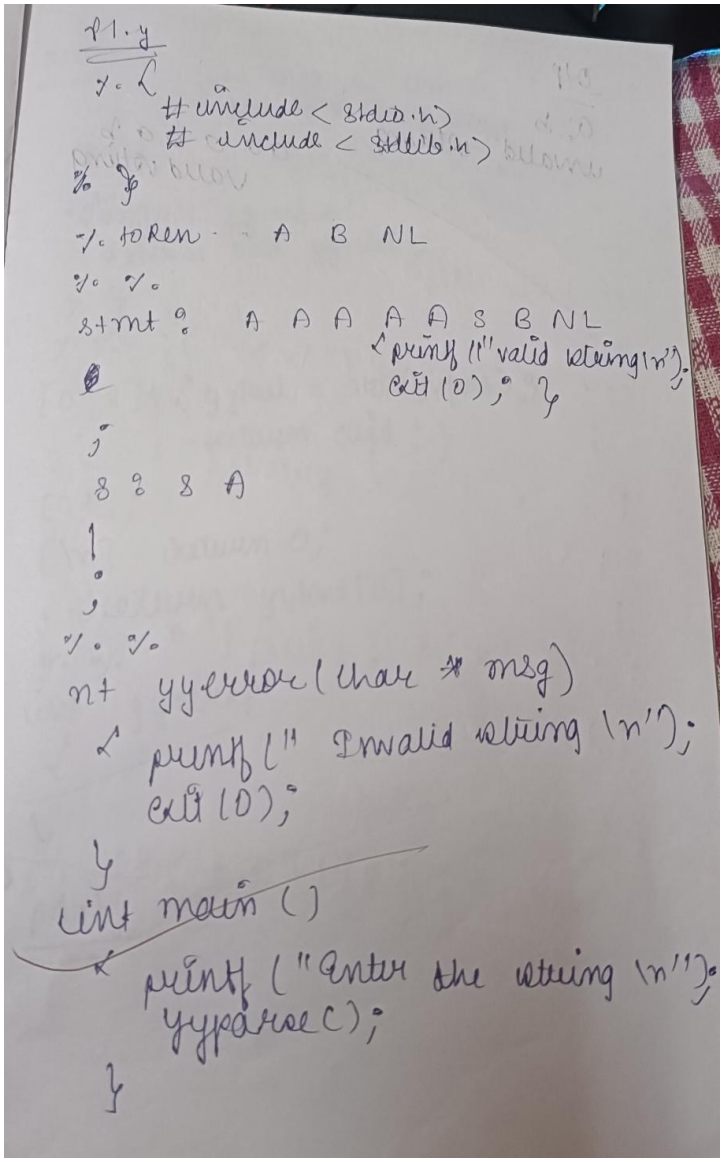
Output


Enter an arithmetic expression:
2+3*4
Valid expression!
Result:14
Enter an arithmetic expression:
2++3-
Invalid expression!

7.2 Write a program in YACC to recognize strings of the form {(a^n)b , n>=5}.

Code



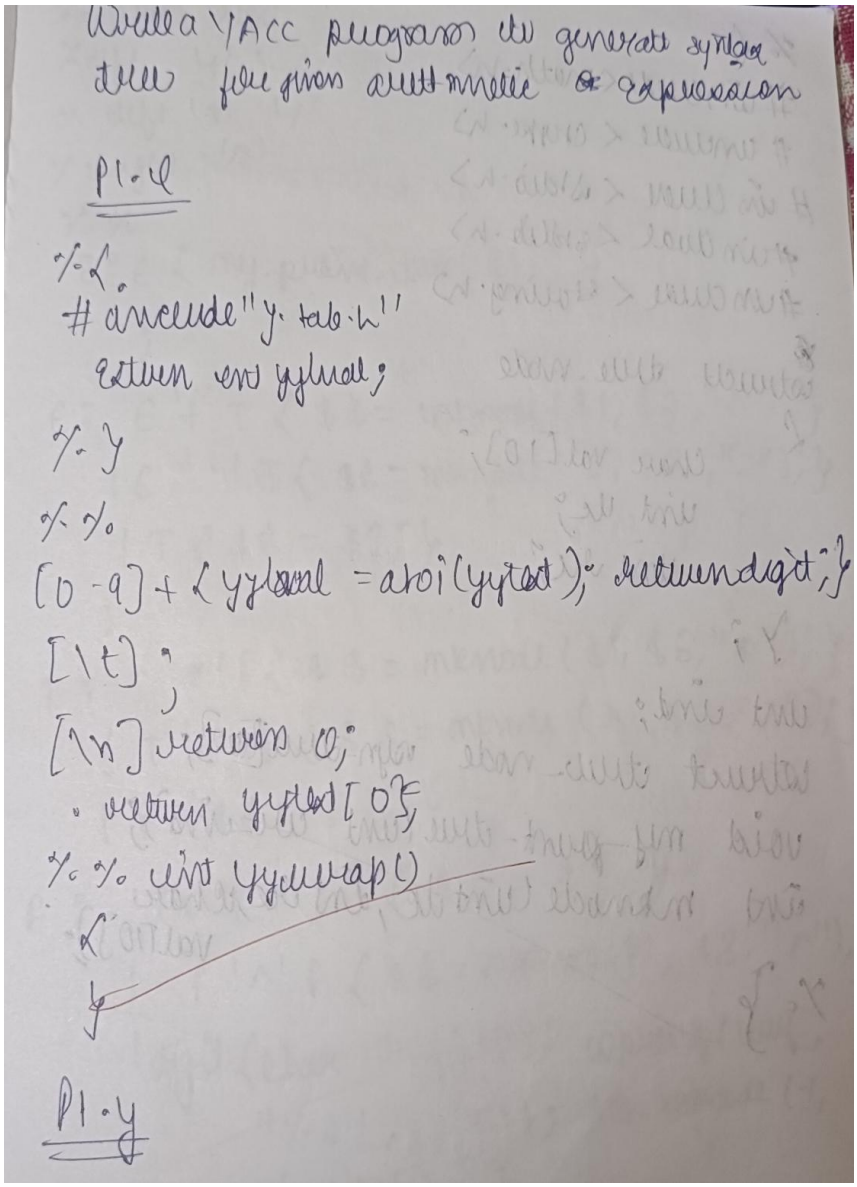


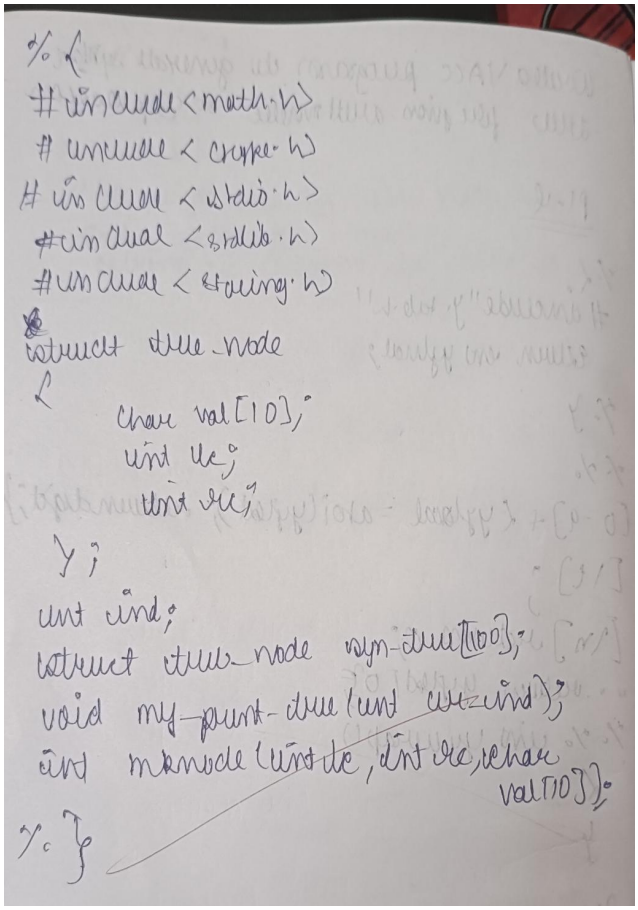
Output

```
Enter a string!
aaaaaaab
Parsed using the rule (a^n)b, n>=5.
Valid String!
ab
Invalid String!


Enter a string!
abc
Invalid String!
```

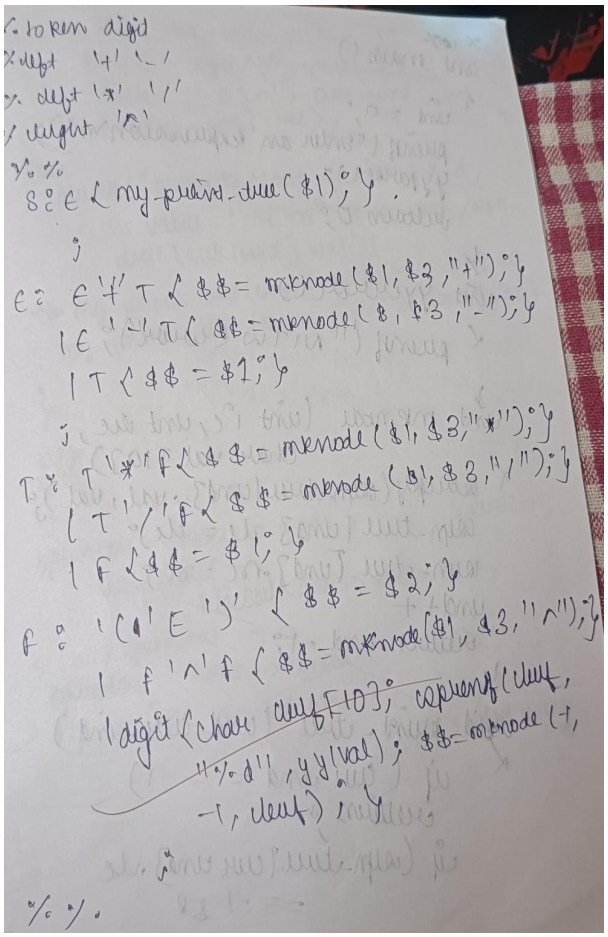
7.3 Write a program in YACC to generate syntax tree for a given arithmetic expression.

Code

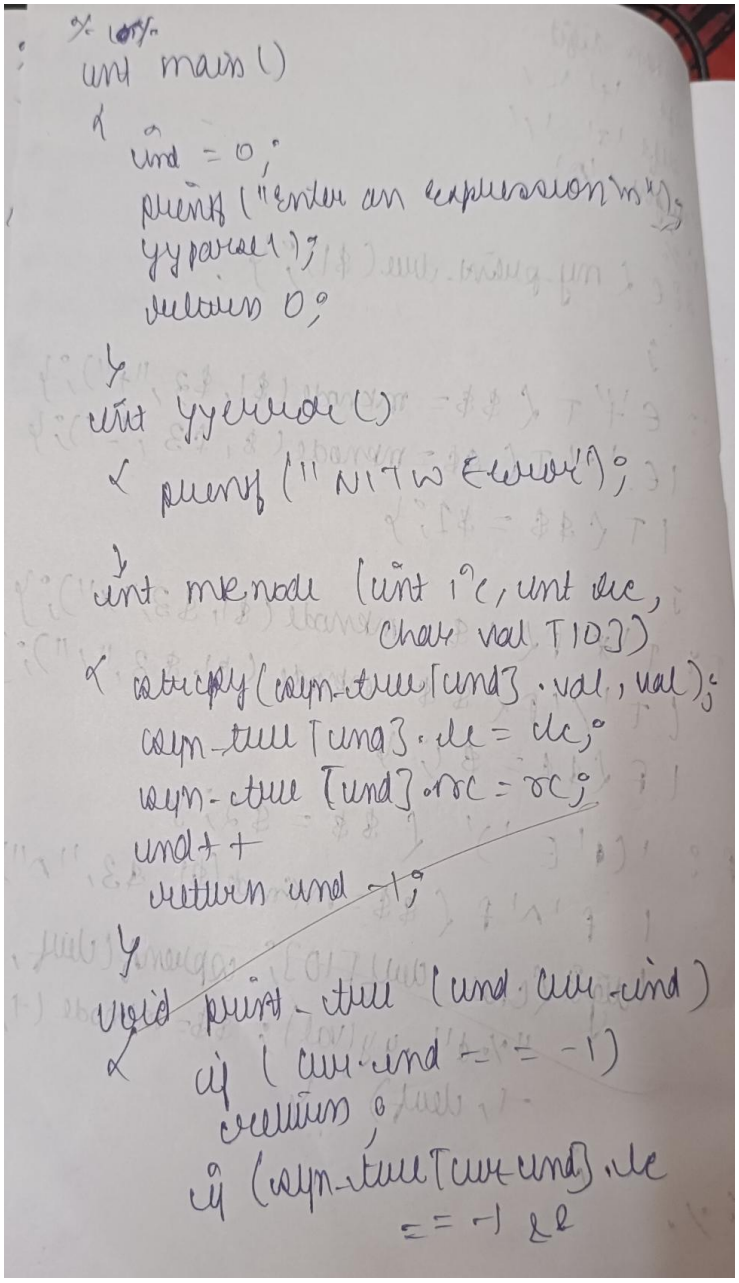




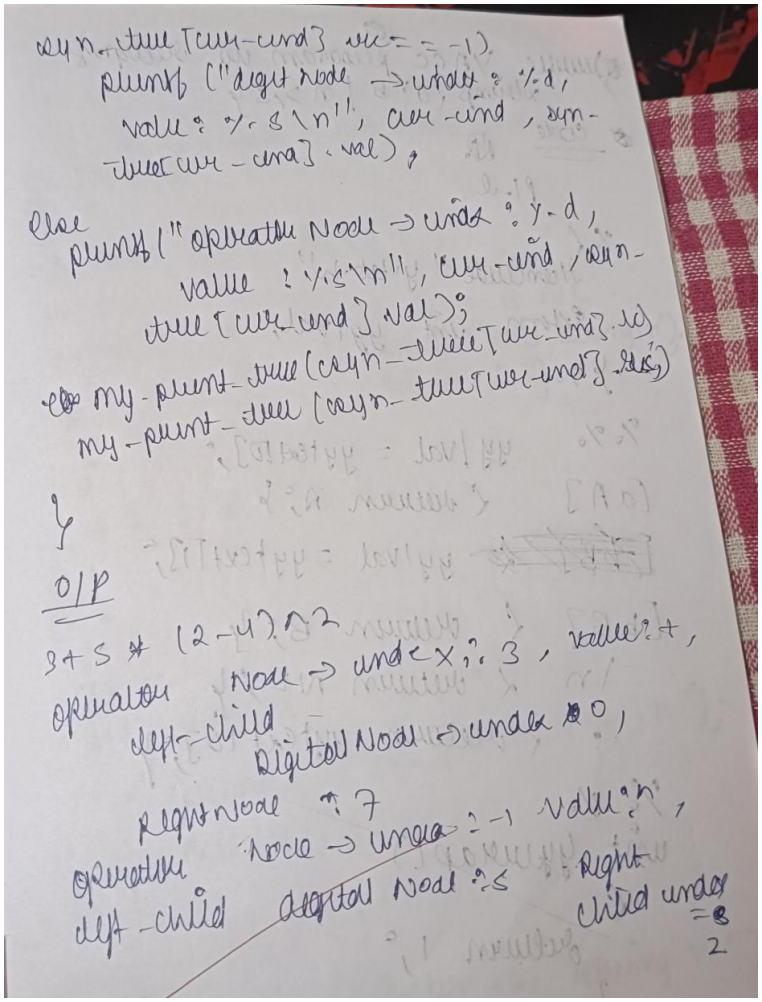
```
%{
#include<math.h>
#include<ctype.h>
#include<stdio.h>
#include<stdlib.h>
#include<string.h>

struct tree_node
{
    char val[10];
    int le;
    int ri;
};
int ind;
struct tree_node syn_tree[100];
void my_print_tree(int cur_ind);
int mknode(int le, int ri, char val[10]);
%}
```



```
%token digit
%left '+' '-'
%left '*' '/'
%right '^'
%%
S: E { my_print_tree($1); }
 ;
E: E '+' T { $$ = mknode($1, $3, "+"); }
 | E '-' T { $$ = mknode($1, $3, "-"); }
 | T { $$ = $1; }
 ;
T: T '*' F { $$ = mknode($1, $3, "*"); }
 | T '/' F { $$ = mknode($1, $3, "/"); }
 | F { $$ = $1; }
F: '(' E ')' { $$ = $2; }
 | F '^' F { $$ = mknode($1, $3, "^"); }
 | digit { char buff[10]; sprintf(buff,
     "%d", yylval); $$ = mknode(-1,
     -1, buff); }
 ;
%%
```



```
%{
int main()
{
    ind = 0;
    printf("Enter an expression\n");
    yyparse();
    return 0;
}
int yyerror()
{
    printf("NITW Error");
}
int mknode(int lc, int rc,
           char val[10])
{
    strcpy(syn_tree[ind].val, val);
    syn_tree[ind].lc = lc;
    syn_tree[ind].rc = rc;
    ind++
    return ind-1;
}
void print_tree(ind cur_ind)
{
    if (cur_ind == -1)
    return;
    if (syn_tree[cur_ind].lc
        == -1 &&
```



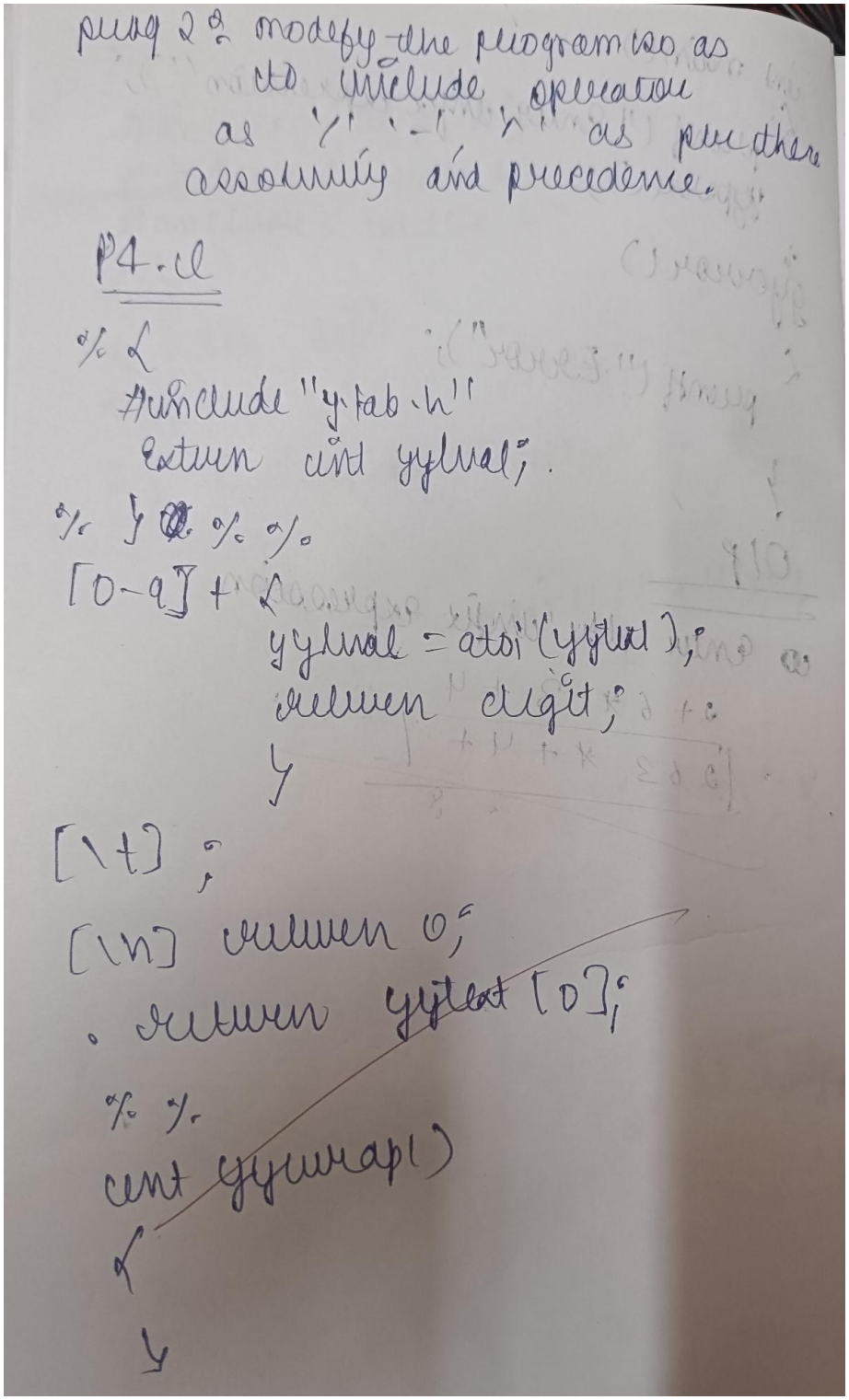
Output

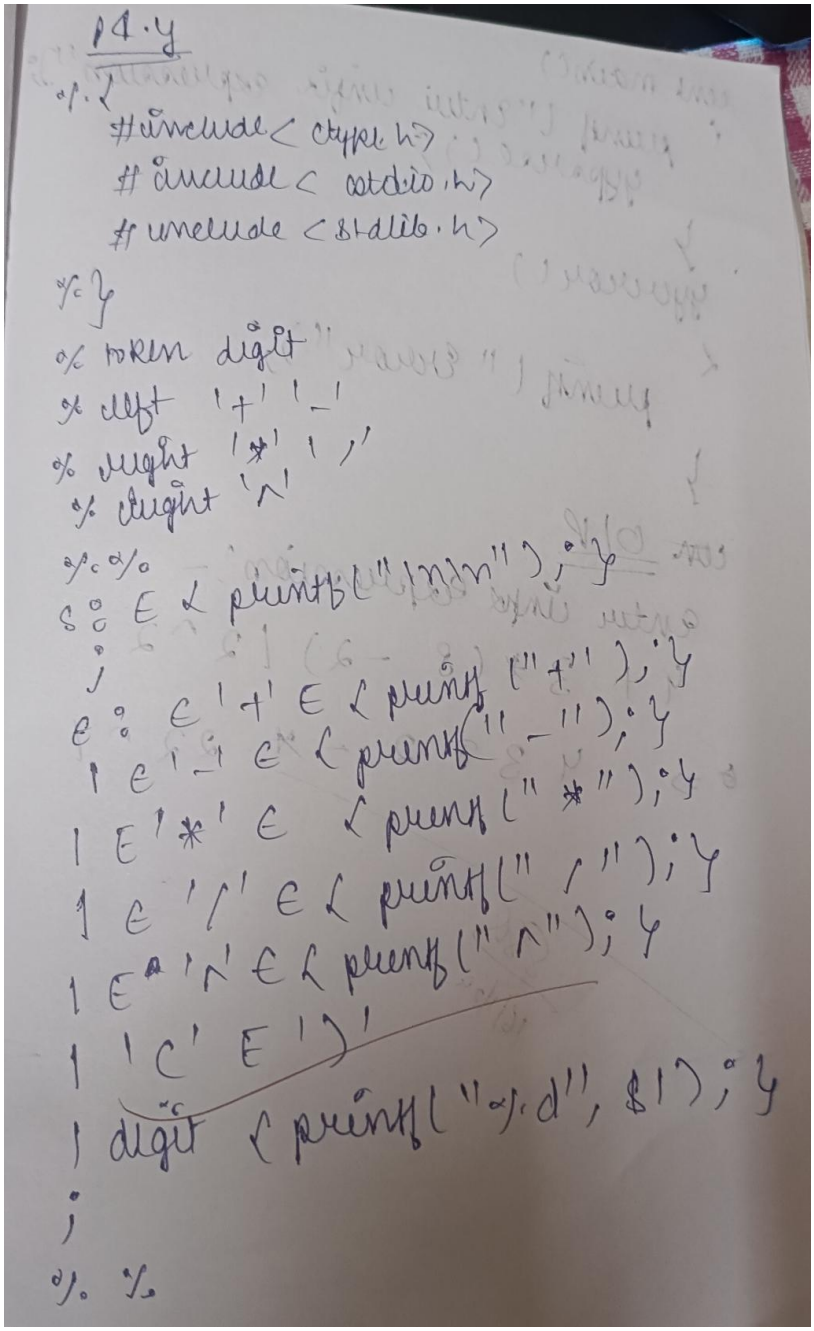
```
Enter an expression:
2*3+5*4
Operator Node -> Index : 6, Value : +, Left Child Index : 2,Right Child Index : 5
Operator Node -> Index : 2, Value : *, Left Child Index : 0,Right Child Index : 1
Digit Node -> Index : 0, Value : 2
Digit Node -> Index : 1, Value : 3
Operator Node -> Index : 5, Value : *, Left Child Index : 3,Right Child Index : 4
Digit Node -> Index : 3, Value : 5
Digit Node -> Index : 4, Value : 4
```

Lab 8

8.1 Write a program in YACC to convert infix to postfix expression.

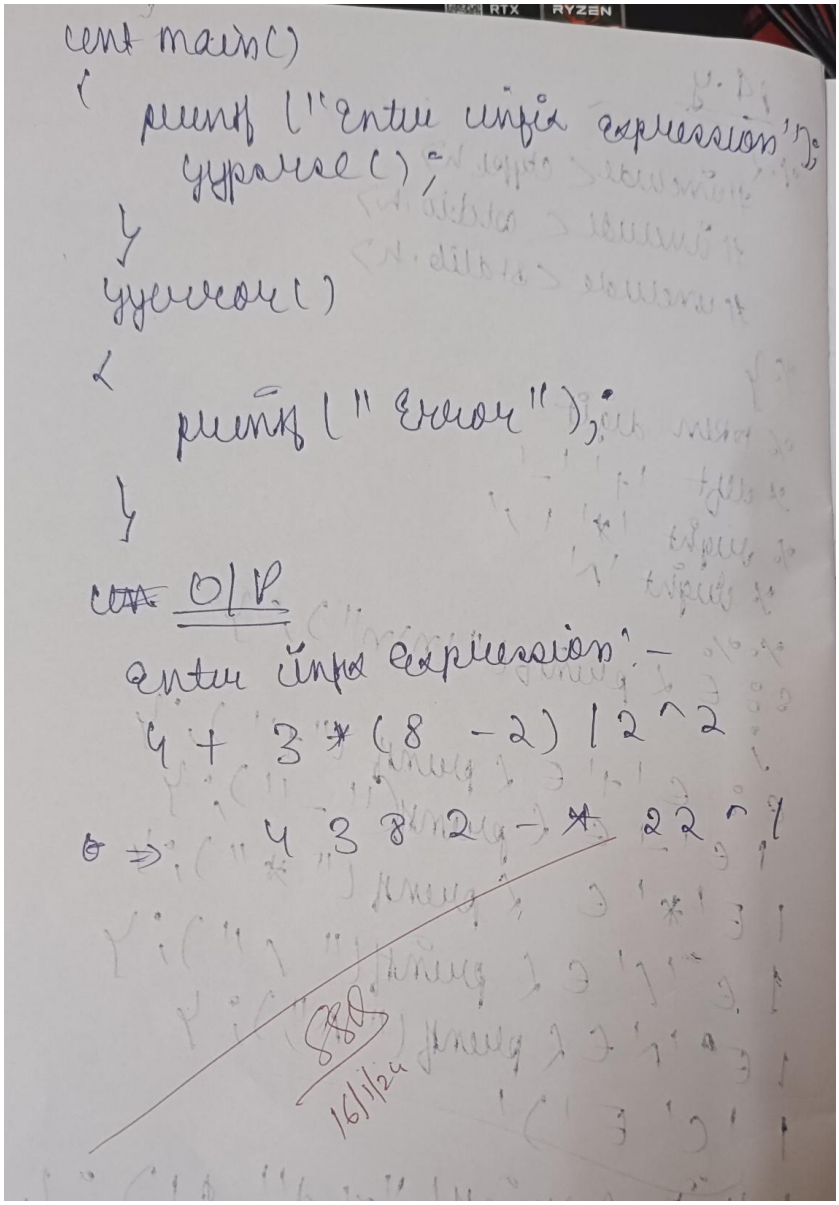
Code





14.y

```
%{
#include <ctype.h>
#include <stdio.h>
#include <stdlib.h>
%}
%token digit
%left '+' '-'
%left '*' '/'
%right '^'
%%
S : E { printf("\n\n"); }
  ;
E : E '+' E { printf("+"); }
  | E '-' E { printf("-"); }
  | E '*' E { printf("*"); }
  | E '/' E { printf("/"); }
  | E '^' E { printf("^"); }
  | '(' E ')'
  | digit { printf("%d", $1); }
  ;
%%
```



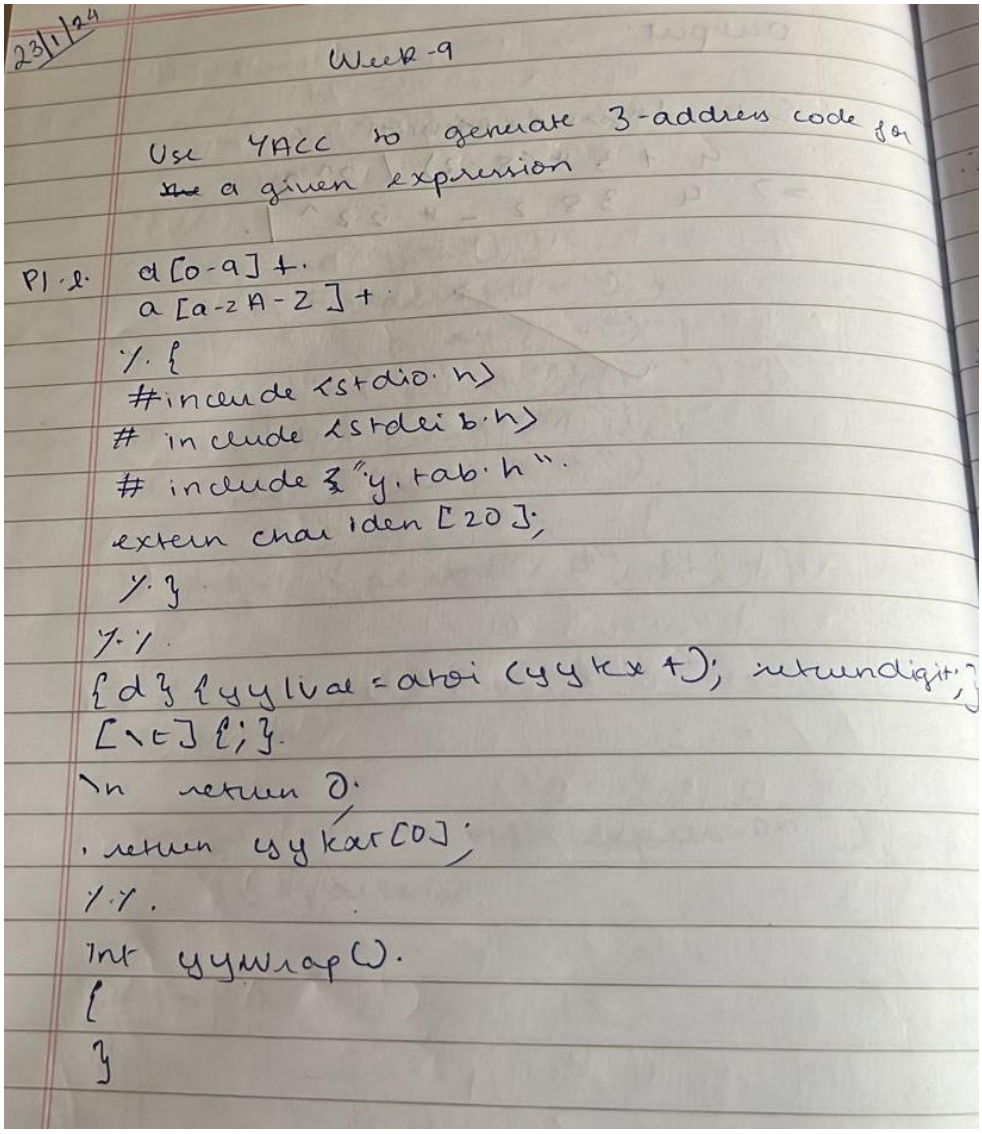
Output

```
Enter an infix expression:
2+3*8/4^3-3
238*43^/+3-
```

Lab 9

9.1 Write a program in YACC to generate three address code for a given expression.

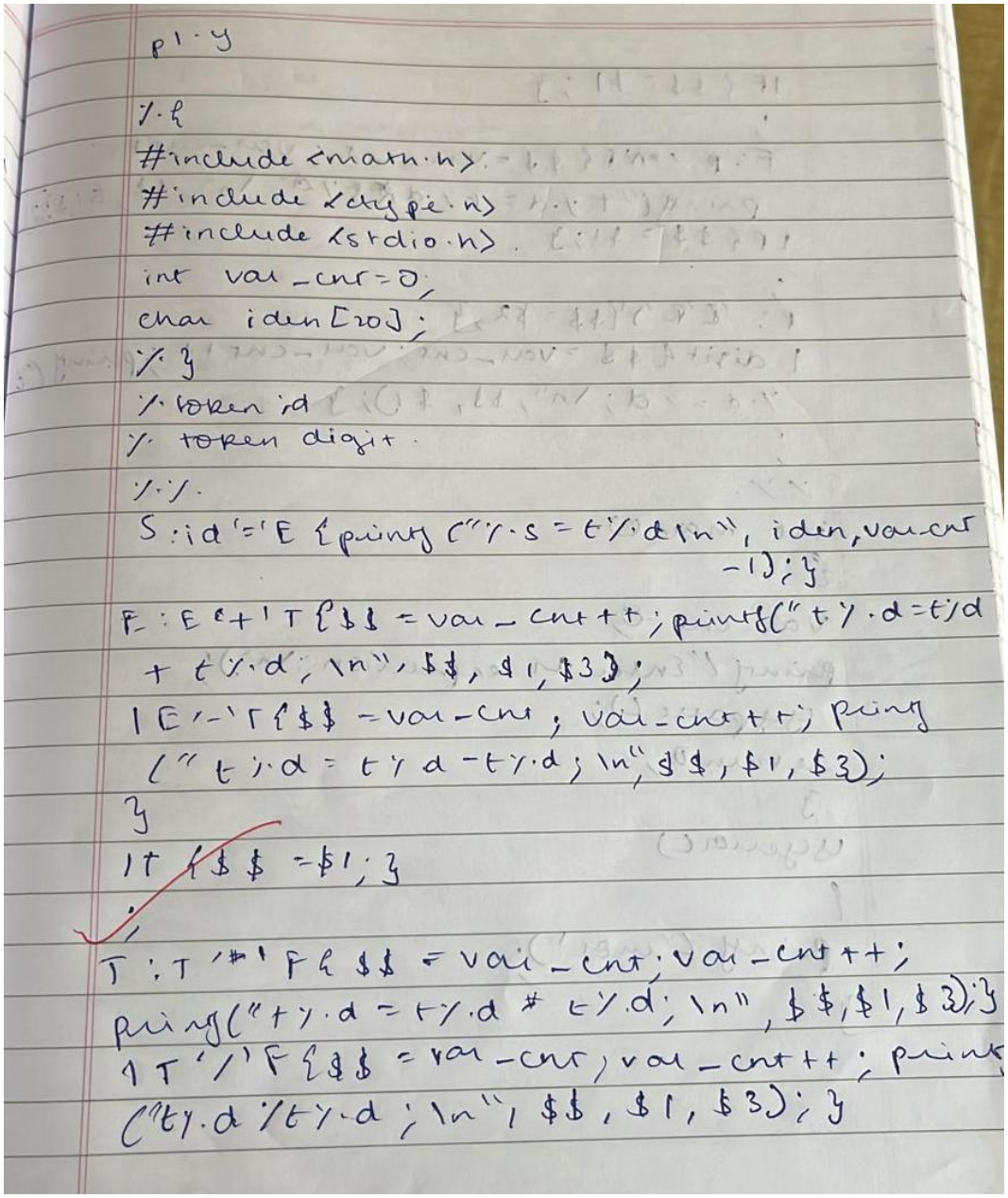
Code


23/1/24

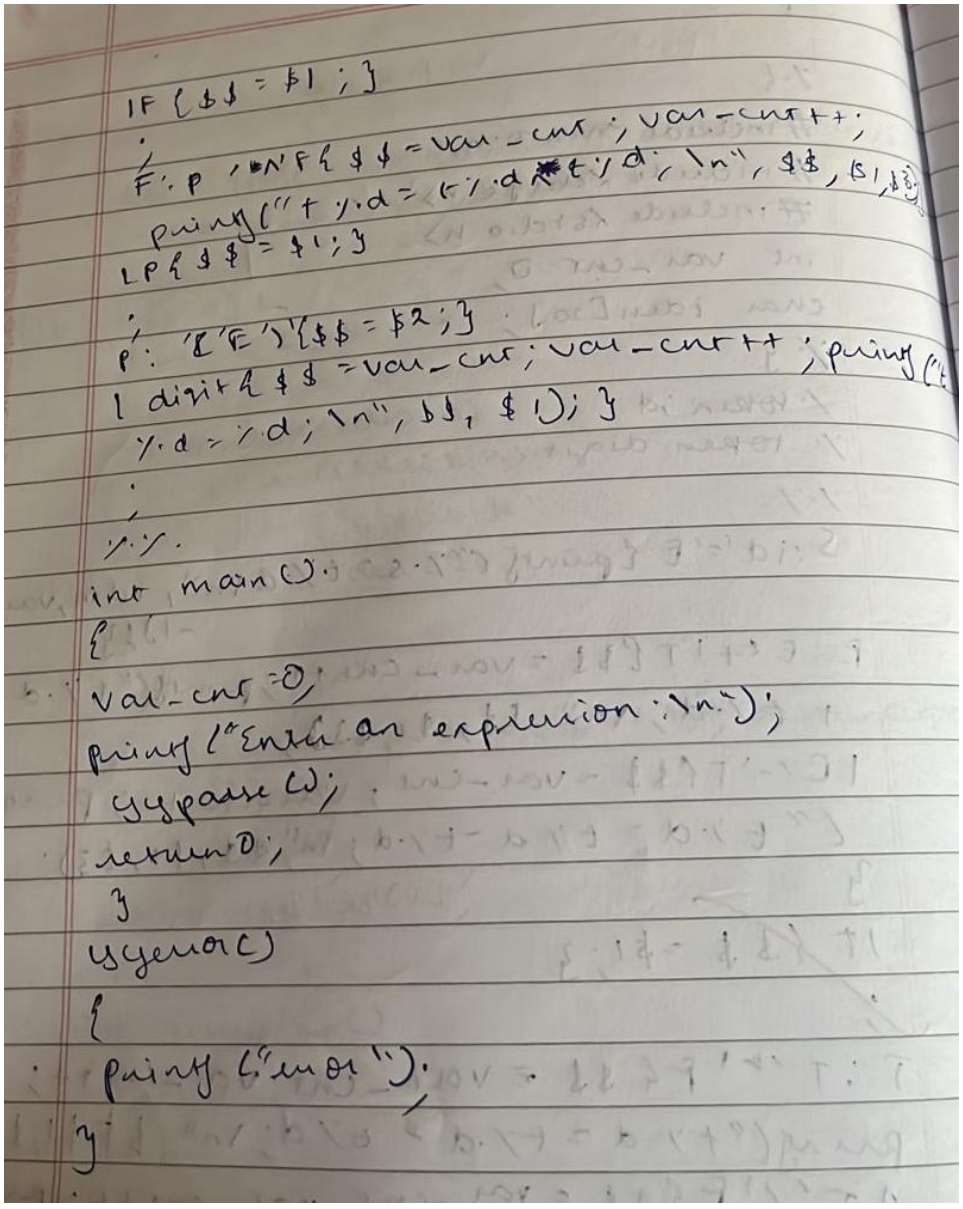
Week-9

Use YACC to generate 3-address code for a given expression

```
Pl.l.  d[0-9]+.
       a[a-zA-Z]+
%{
#include<stdio.h>
#include<stdlib.h>
#include"y.tab.h".
extern char iden[20];
%}
%%
{d}{yylval=atoi(yytext); return digit;}
[\t]{;}.
\n return 0.
. return yytext[0];
%%.
int yywrap().
{
}
```



```
p1.y

%{
#include <math.h>
#include <ctype.h>
#include <stdio.h>
int var_cnt=0;
char iden[20];
%}
%token id
%token digit
%%
S:id'='E {printf("%s = t%d\n", iden, var_cnt
                                    -1);}

E: E'+'T {$$ = var_cnt++; printf("t%d = t%d
 + t%d\n", $$, $1, $3);
|E'-'T {$$ = var_cnt; var_cnt++; printf
 ("t%d = t%d - t%d\n", $$, $1, $3);
}
|T {$$ =$1;}
;
T: T'*'F {$$ = var_cnt; var_cnt++;
printf("t%d = t%d * t%d\n", $$,$1,$3);}
|T'/'F {$$ = var_cnt; var_cnt++; printf
("t%d = t%d / t%d\n", $$, $1, $3);}
```



```
IF [$$ = $1 ; ]
;
F: P '+' F { $$ = var_cnt ; var_cnt++;
 printf("t%d = t%d + t%d; \n", $$, $1, $3);}
LP { $$ = $1; }
;
P: '(' E ')' { $$ = $2; }
| digit { $$ = var_cnt; var_cnt++ ; printf("t
 %d = %d; \n", $$, $1); }
;
%%
int main().
{
var_cnt = 0;
printf("Enter an expression: \n");
yyparse();
return 0;
}
yyerror()
{
printf("error");
}
```

Output

Enter an expression.

a = 2 + 3 * 6.

t0 = 2;

t1 = 3;

t2 = 6;

t3 = t1 * t2;

t4 = t0 + t3;

a = t4.

Output


Enter an expression:
a=2*3/6-4
t0 = 2;
t1 = 3;
t2 = t0 * t1;
t3 = 6;
t4 = t2 / t3;
t5 = 4;
t6 = t4 - t5;
a=t6