VISVESVARAYA TECHNOLOGICAL UNIVERSITY

"JnanaSangama", Belgaum -590014, Karnataka.

LAB REPORT

on

COMPILER DESIGN

Submitted by

Krishn Maloo (1BM21CS092)

Under the Guidance of
Prof. Sunayana S
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 **Krishn Maloo (1BM21CS092)** , 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. Sunayana S	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, Krishn Maloo (1BM21CS092), 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 the guidance of Prof. Sunayana S, 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.

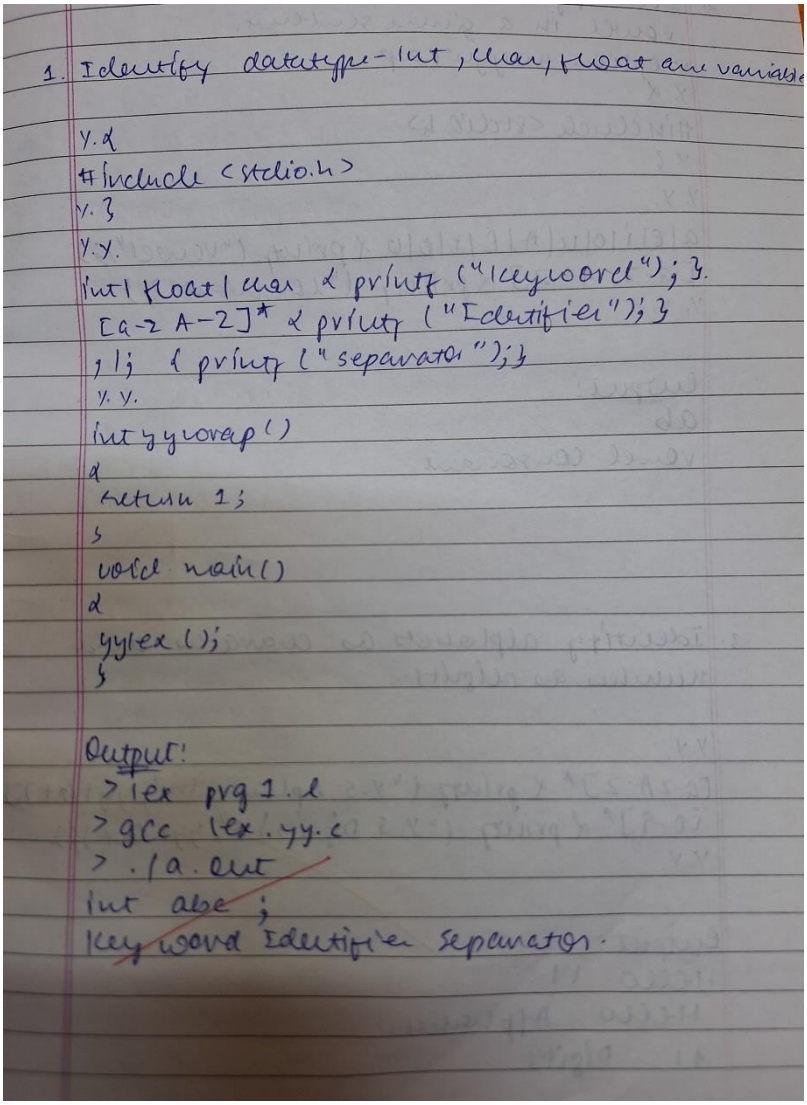
TABLE OF CONTENTS

4.2.1	The set of all string ending in 00.
4.2.2	The set of all strings with three consecutive 222's.
4.2.3	The set of all string such that every block of five consecutive symbols contains at least two 5's.
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.
4.2.5	The set of all strings such that the 10th symbol from the right end is 1.
4.2.6	The set of all four digits numbers whose sum is 9.
4.2.7	The set of all four digital numbers, whose individual digits are in ascending order from left to right.
5	
5.1	Write a C program to design lexical analysis to recognize any five keywords, identifiers, numbers, operators and punctuations.
6	
6.1	Write a program to perform recursive descent parsing on the following grammar: S->cAd A->ab | a
7	
7.1	Write a program in YACC to design a suitable grammar for evaluation of arithmetic expression having +, -, * and /.
7.2	Write a program in YACC to recognize strings of the form {(a^n)b,n>=5}.
7.3	Write a program in YACC to generate a syntax tree for a given arithmetic expression.
8	
8.1	Write a program in YACC to convert infix to postfix expression.
9	
9.1	Write a program in YACC to generate three address code for a given expression.

Lab 1

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

Code:



1. Identify datatype - int, char, float and variable

```
%{
#include <stdio.h>
%}
%%
int|float|char { printf("keyword"); }.
[a-z A-Z]* { printf("Identifier"); }
;|; { printf("separator"); }
%%
int yywrap()
{
return 1;
}
void main()
{
yylex();
}
```

Output:

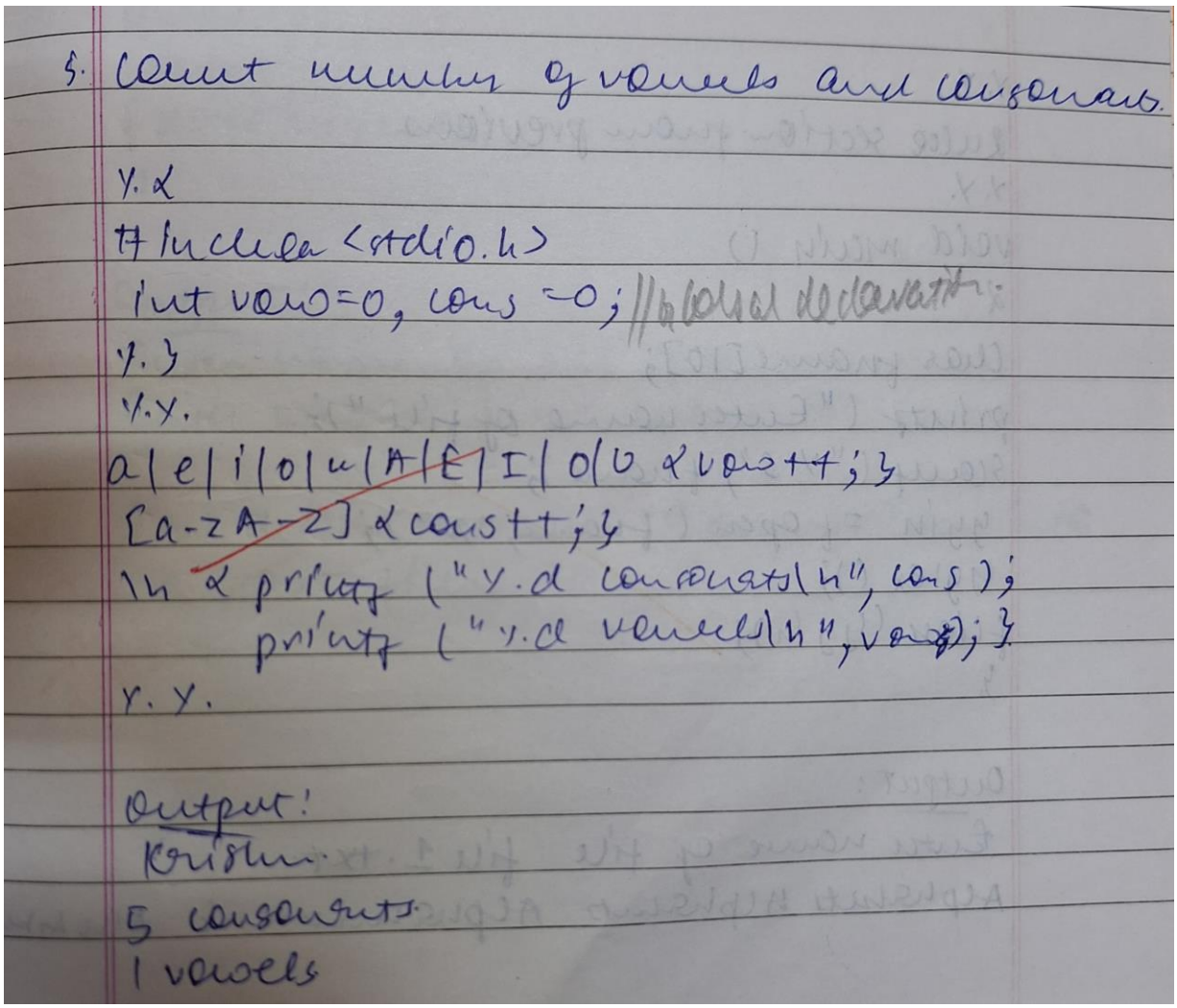
```
> lex prg 1.l
> gcc lex.yy.c
> ./a.out
int abc;
keyword Identifier separator.
```

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 vowels and consonants in a string.

Code



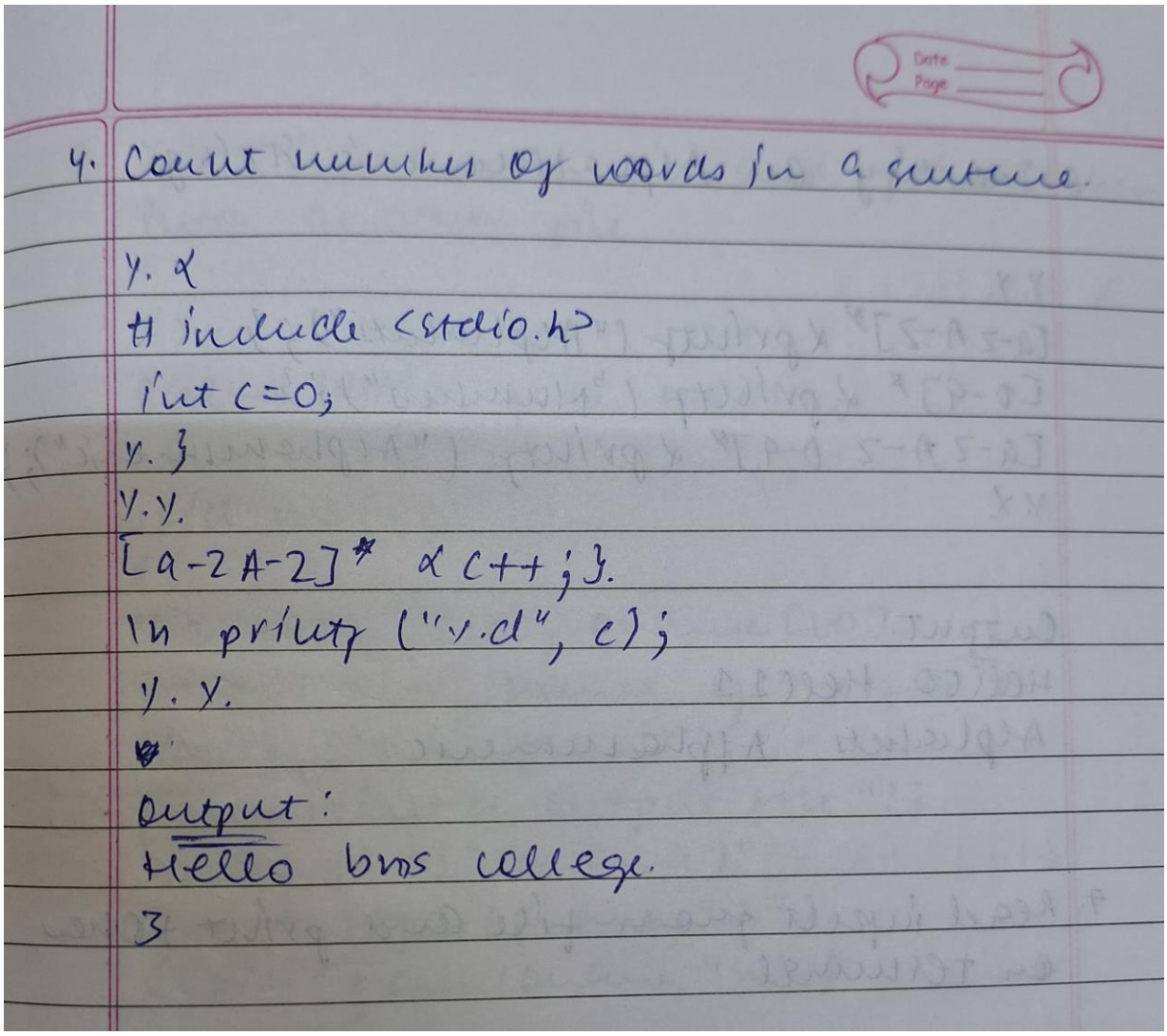
Output

```
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

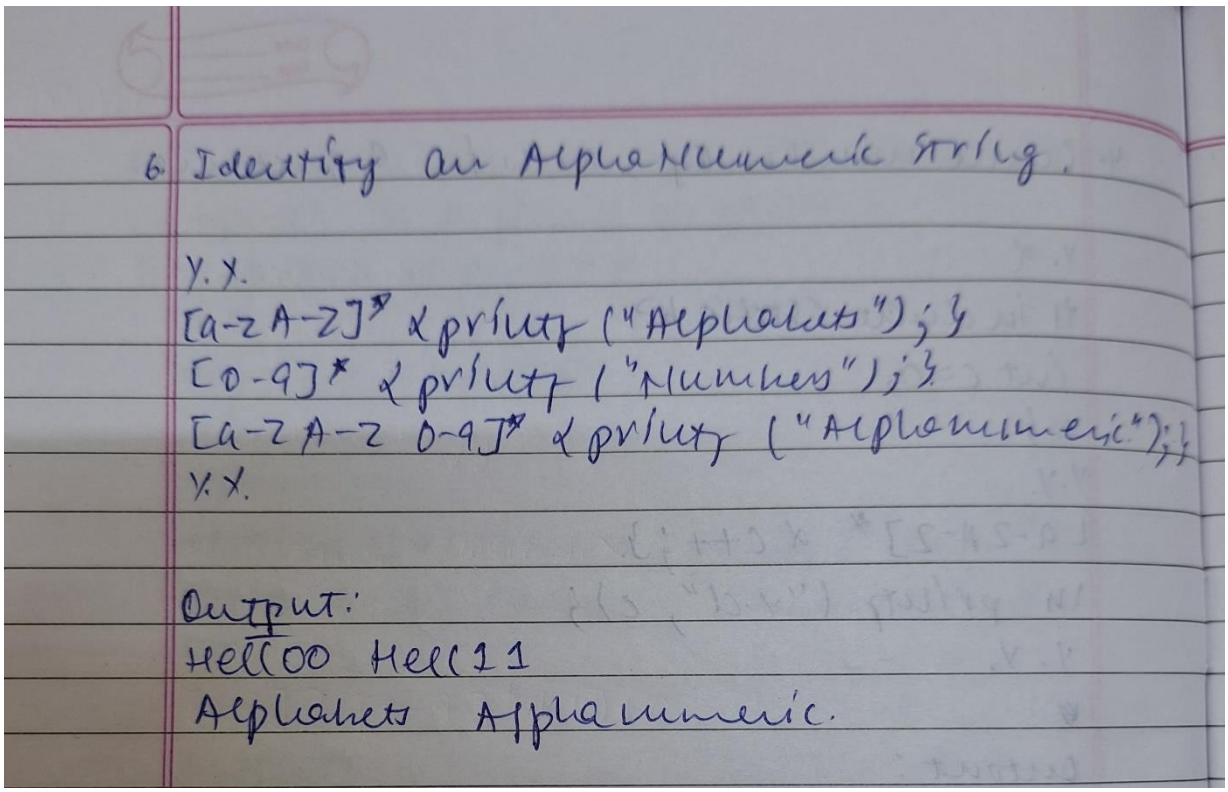


Output

```
Enter a sentence:
This is compiler design lab work.
     No of words in the sentence are 6.
The sun rises in the east and sets in the west.
          No of words in the sentence are 11.
```

2.2 Write a program in lex to demonstrate regular definition.

Code

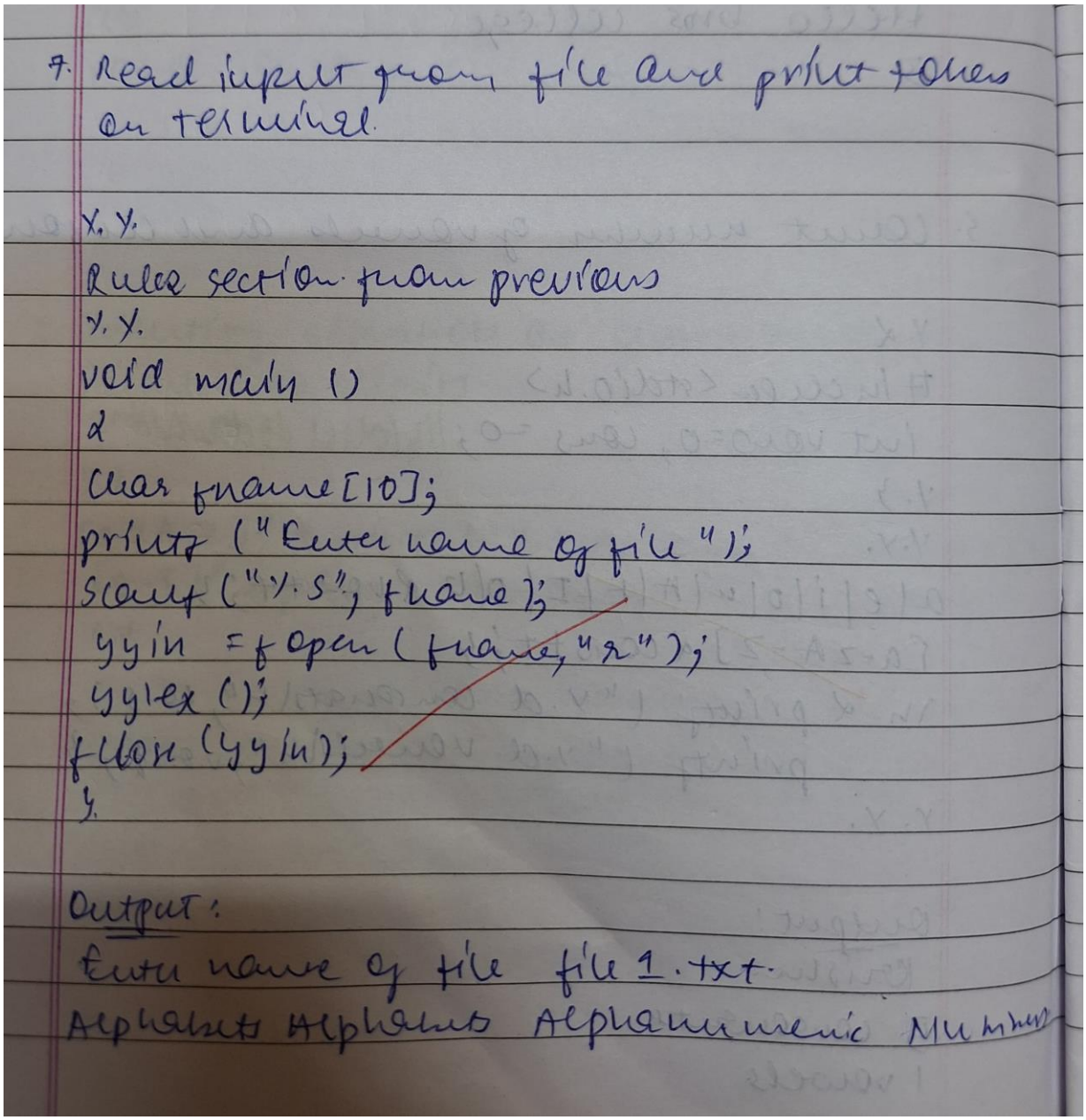


Output

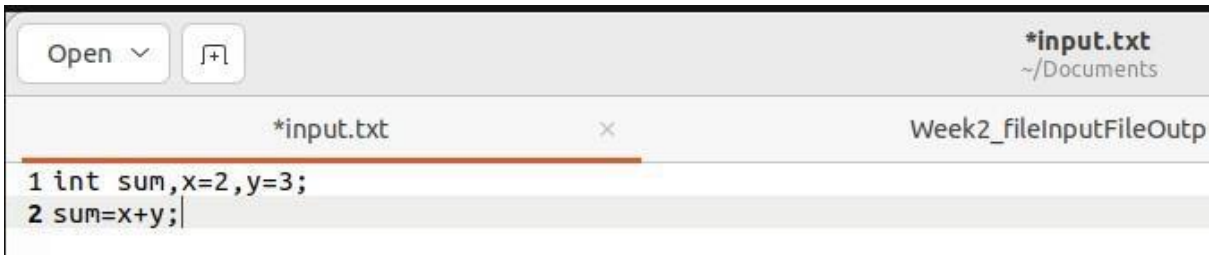
```
Enter a string:
HelloWorld
Characters

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



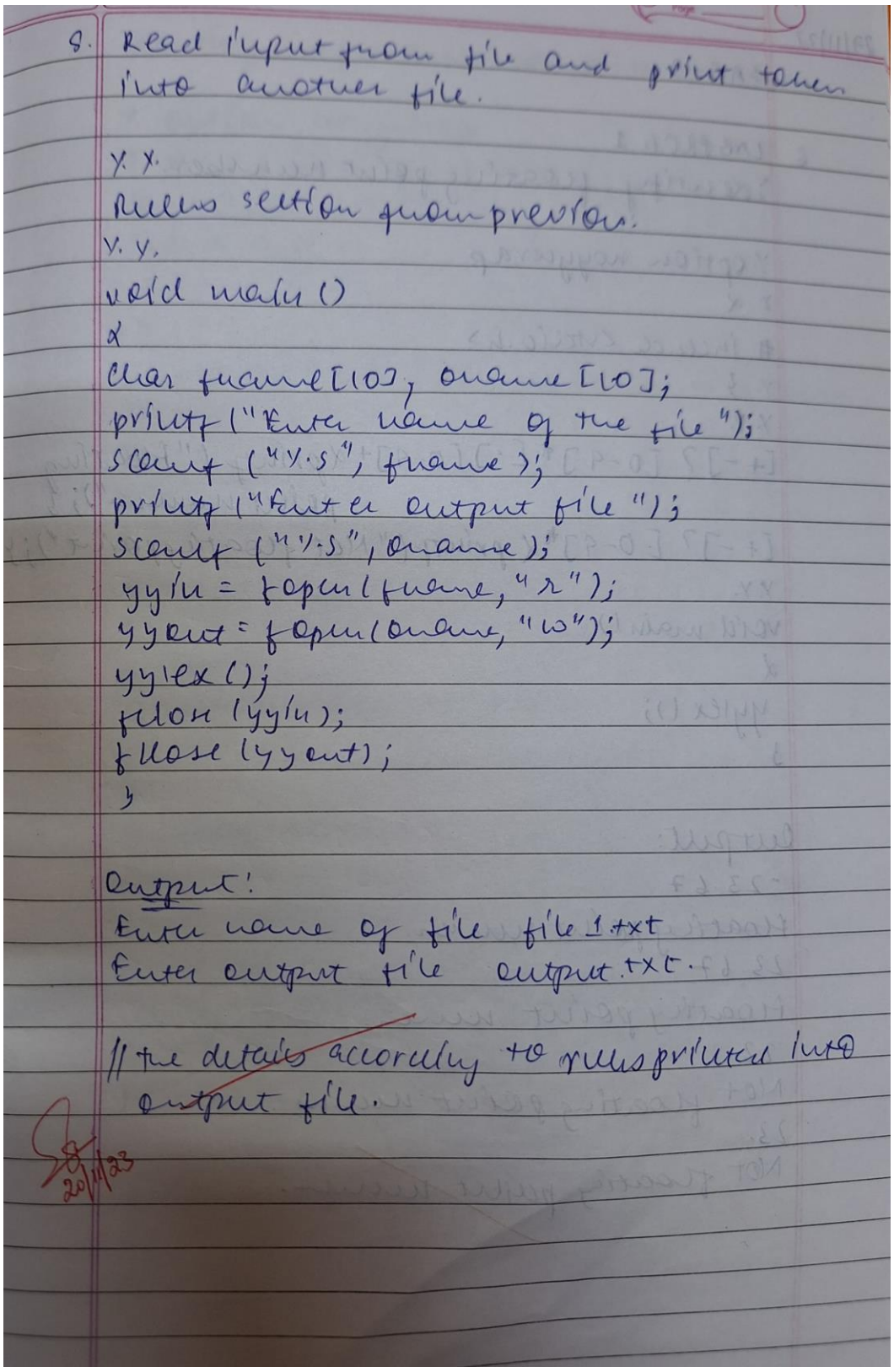
Output




```
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



8. Read input from file and print token into another file.

```
%%
Rules section from previous.
%%
void main()
{
char fname[10], oname[10];
printf("Enter name of the file");
scanf("%s", fname);
printf("Enter output file");
scanf("%s", oname);
yyin = fopen(fname, "r");
yyout = fopen(oname, "w");
yylex();
fclose(yyin);
fclose(yyout);
}
```

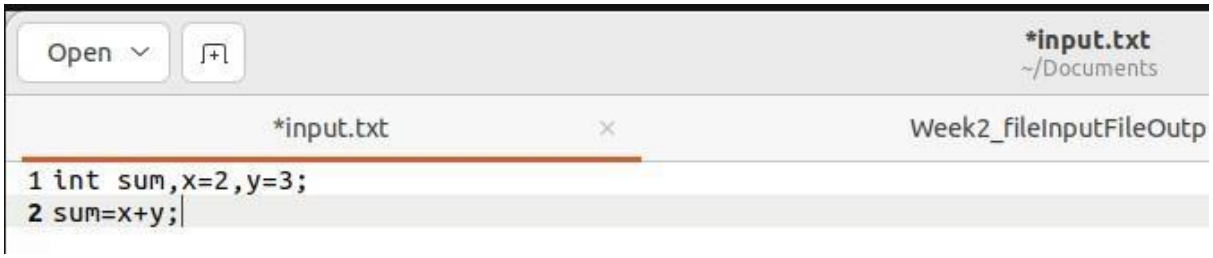
Output:

Enter name of file file1.txt
Enter output file output.txt.

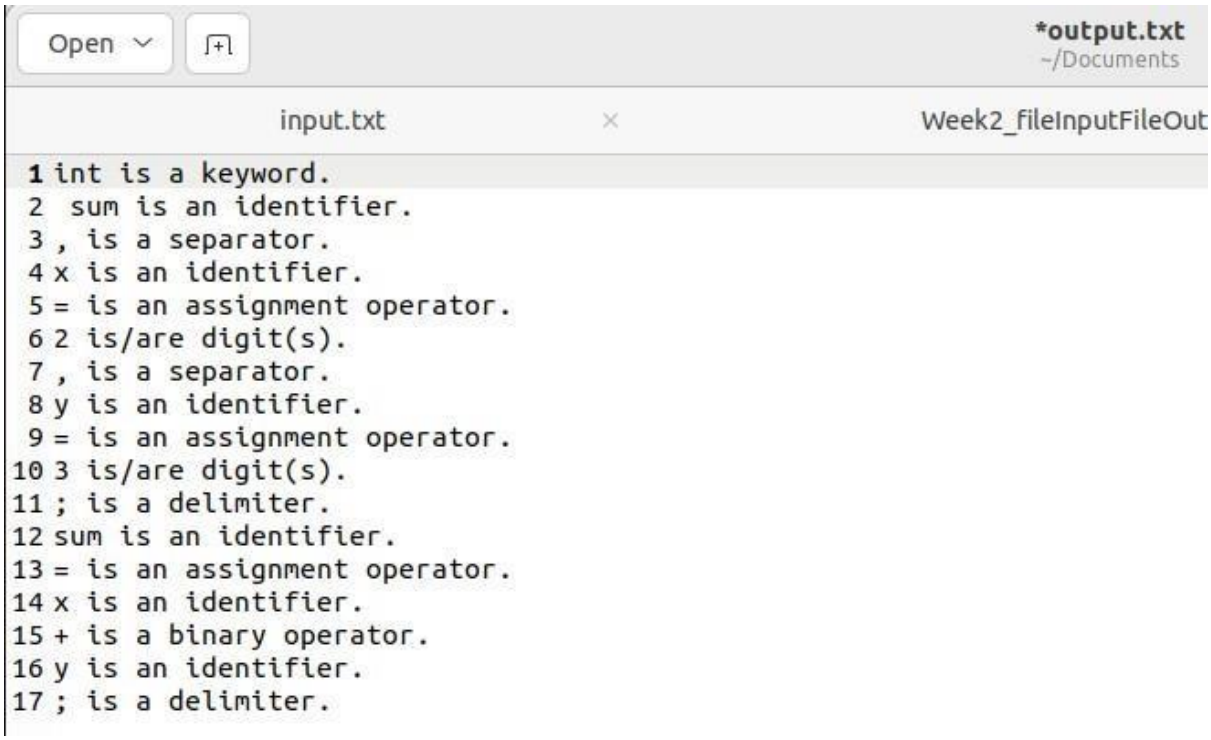
// the details accordingly to rules printed into output file.

20/11/23

Output



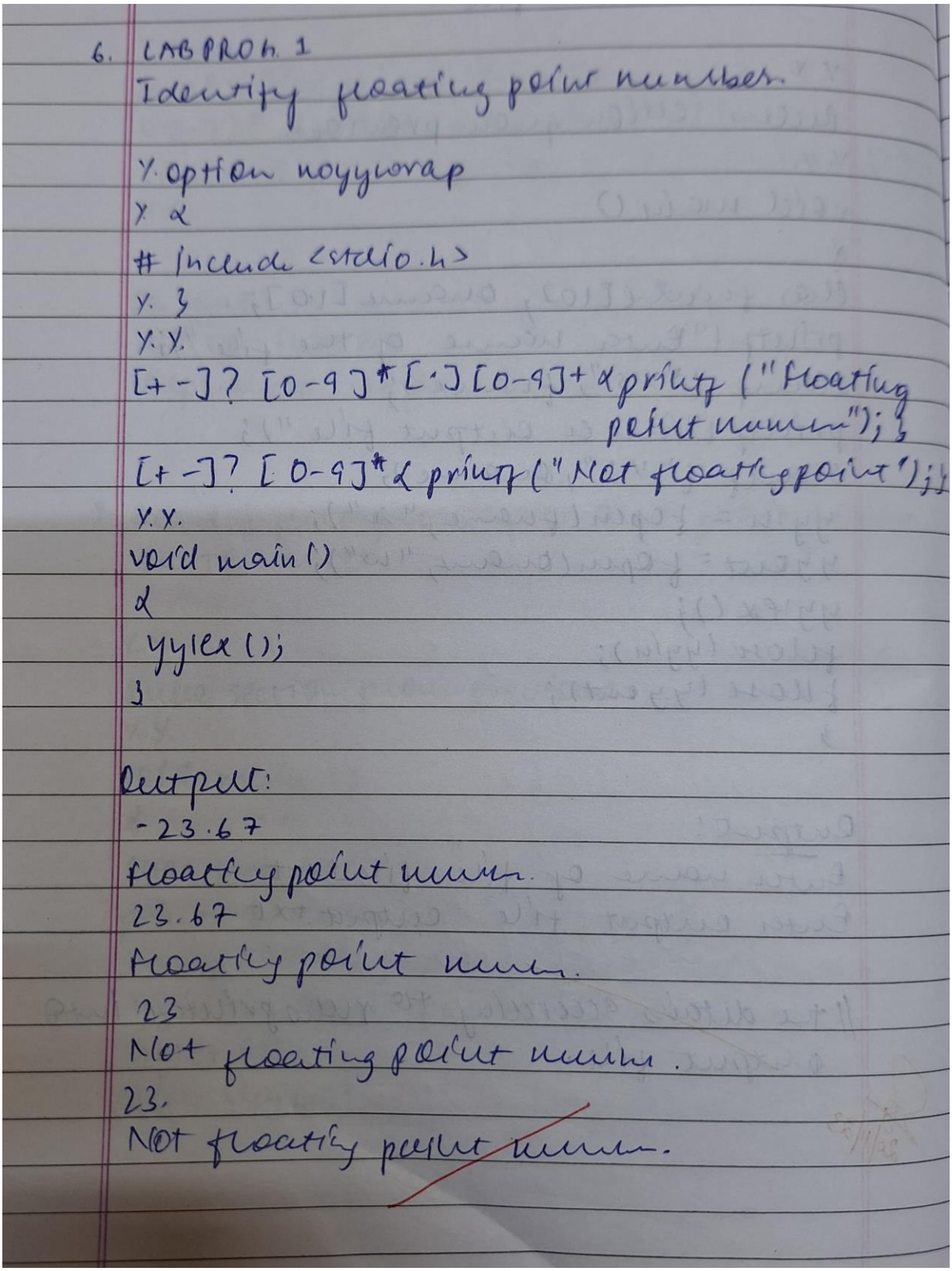

```
Printed in output.txt
```



Lab 3

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

Code



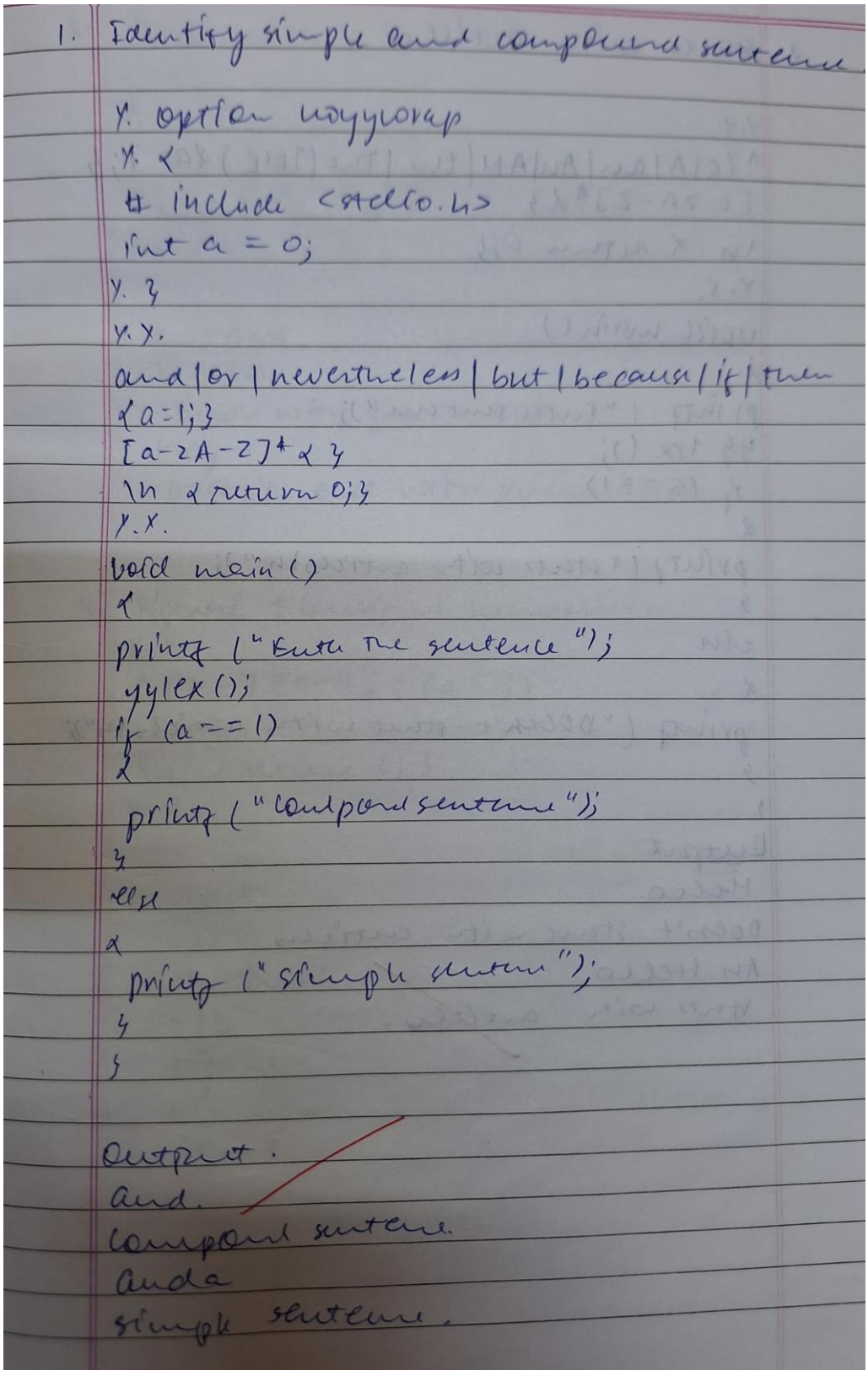
```
6. LAB PROB. 1
Identify floating point number.

%option noyywrap
%{
#include<stdio.h>
%}
%%
[+-]?[0-9]*[.][0-9]+ {printf("floating point number");}
[+-]?[0-9]* {printf("Not floatingpoint");}
%%
void main()
{
 yylex();
}

Output:
-23.67
floating point number.
23.67
floating point number.
23
Not floating point number.
23.
NOT floating point number.
```

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



```
1. Identify simple and compound sentence.

%{ option noyywrap
%}
# include <stdio.h>
int a = 0;
%}
%%
and | or | nevertheless | but | because | if | then
{a=1;}
[a-zA-Z]* {}
\n {return 0;}
%%
void main()
{
printf("Enter the sentence");
yylex();
if (a==1)
{
printf("compound sentence");
}
else
{
printf("simple sentence");
}
}

Output.
and.
compound sentence.
anda
simple sentence.
```

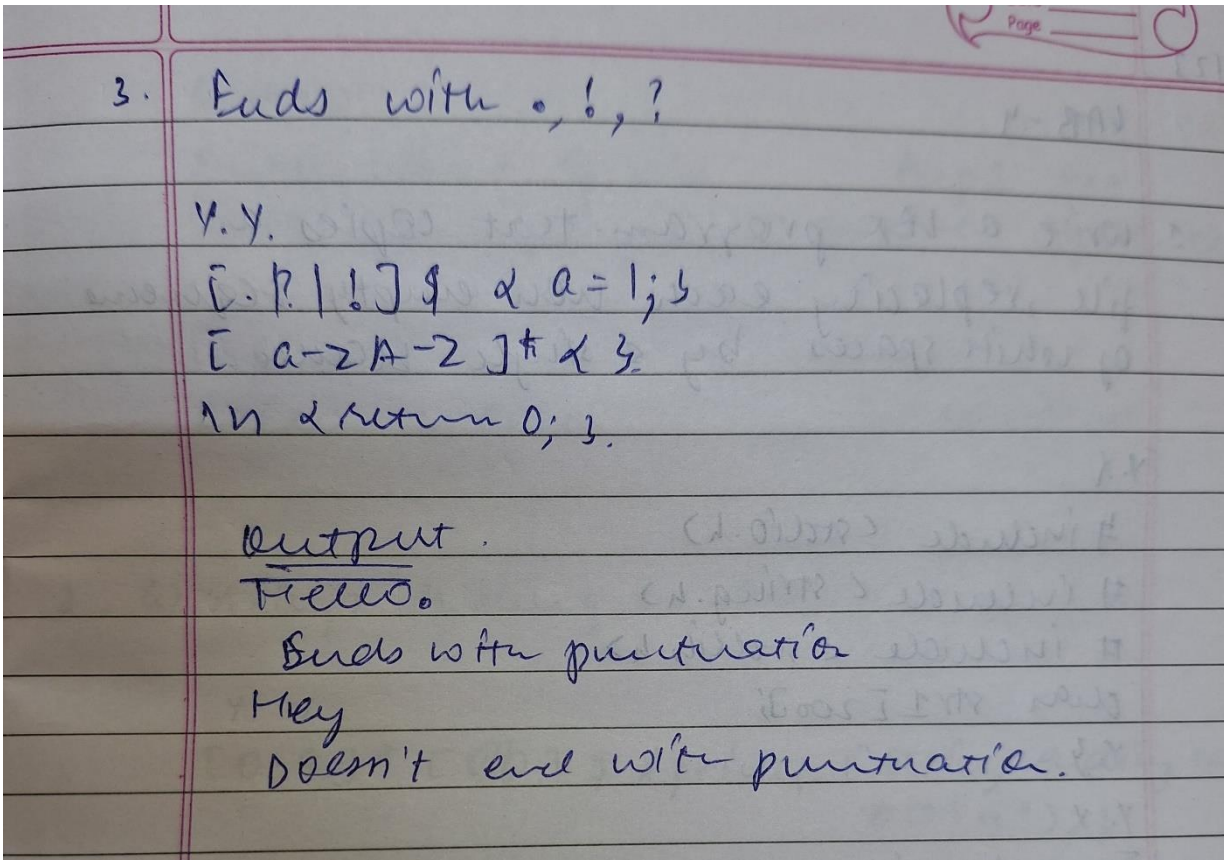
Output

```
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

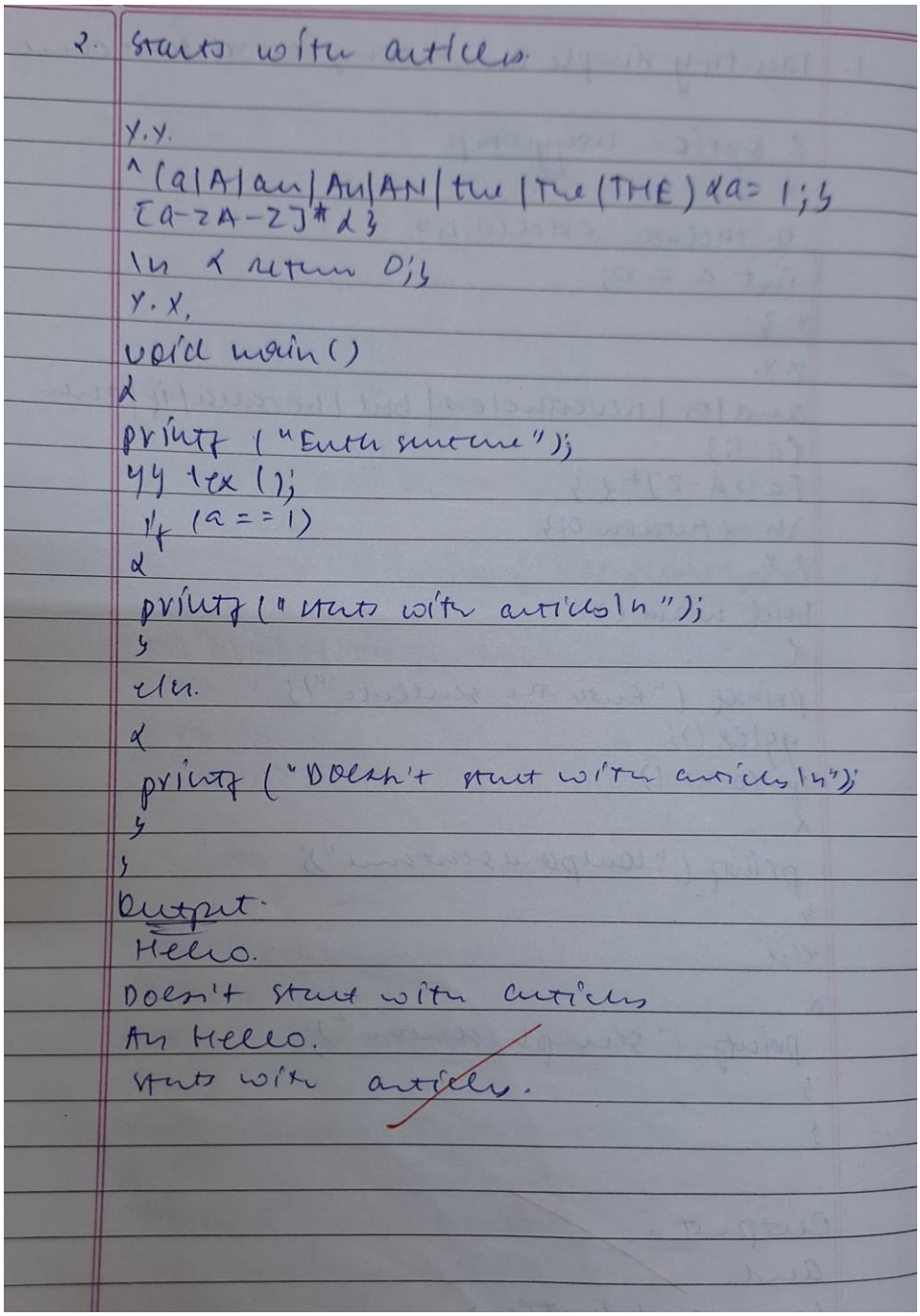
```
Enter a sentence:
Is this yours?
Ends with a punctuation!
```

```
Enter a sentence:
Amazing!
Ends with a punctuation!
```

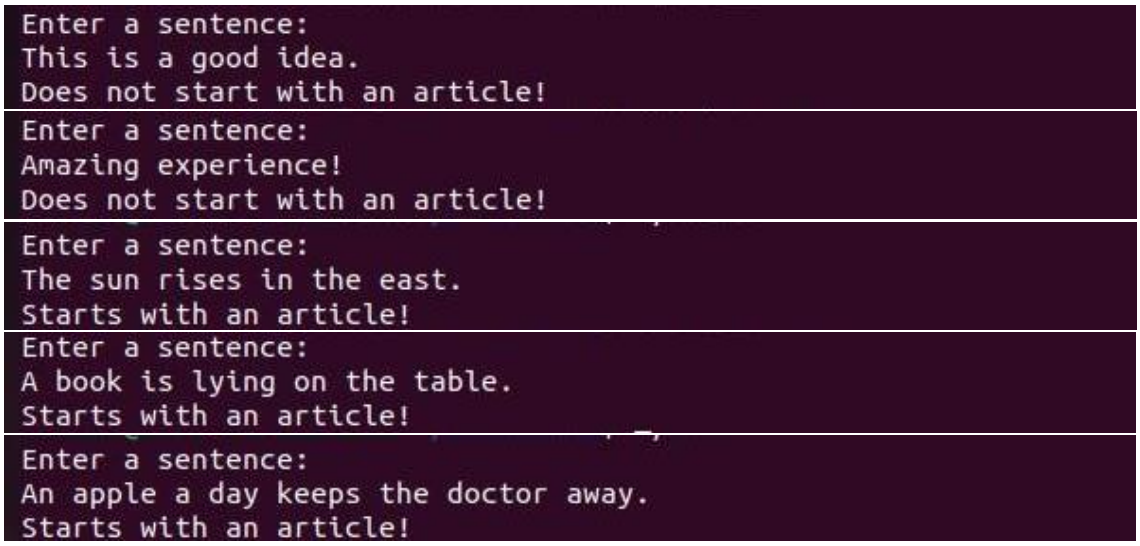
```
Enter a sentence:
You are good
Does not end with punctuation!
```

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

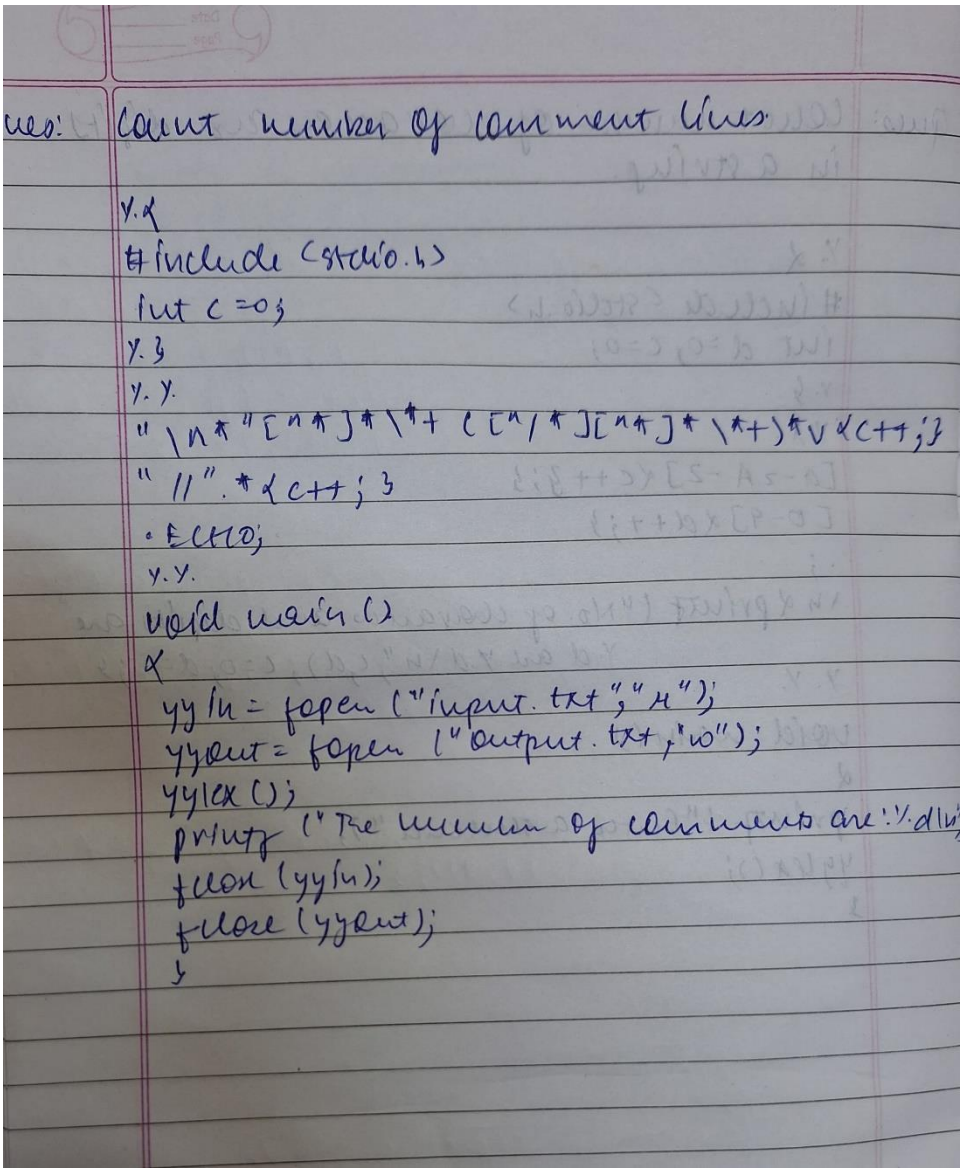


Output


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:
A book is lying on the table.
Starts with an article!
Enter a sentence:
An apple a day keeps the doctor away.
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

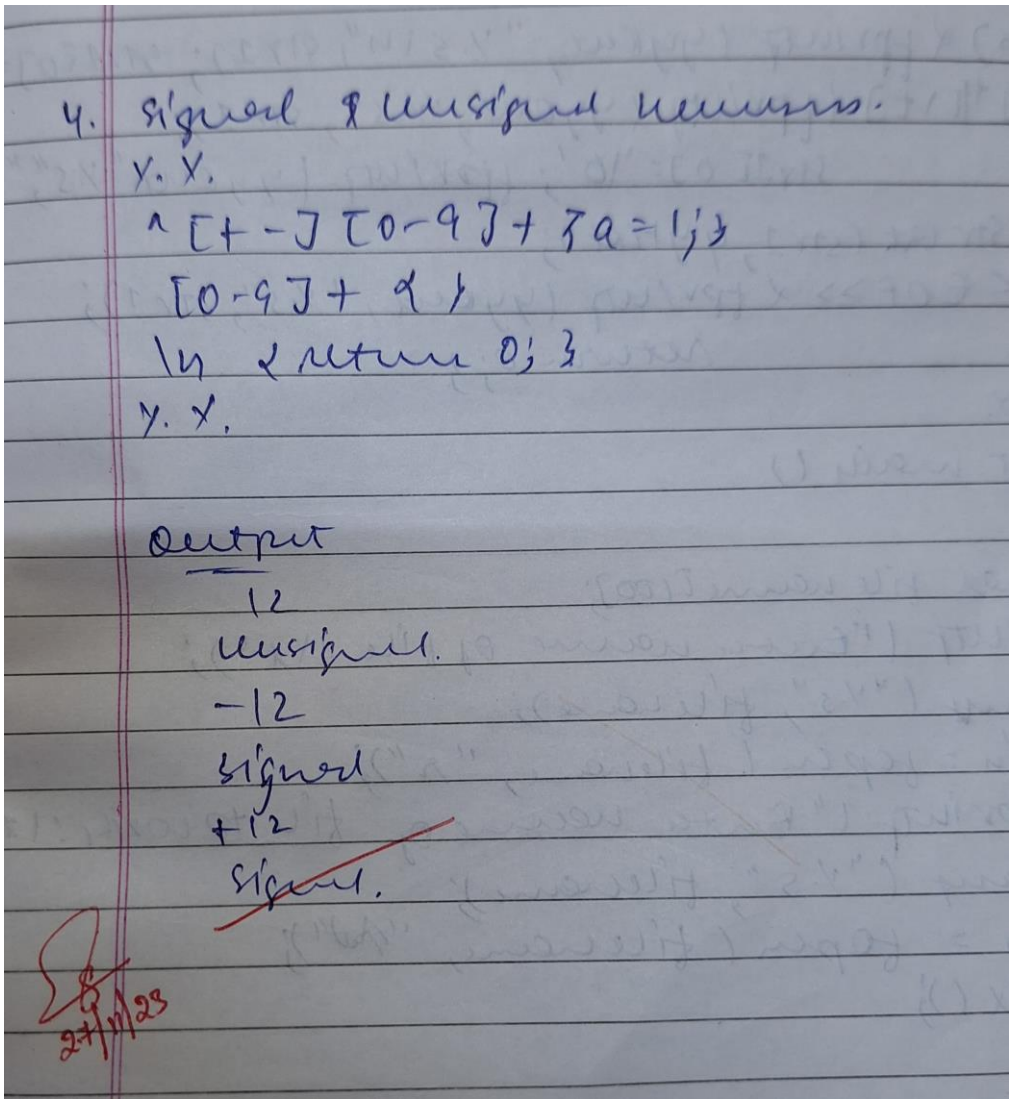


Output

```
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
```

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

Code



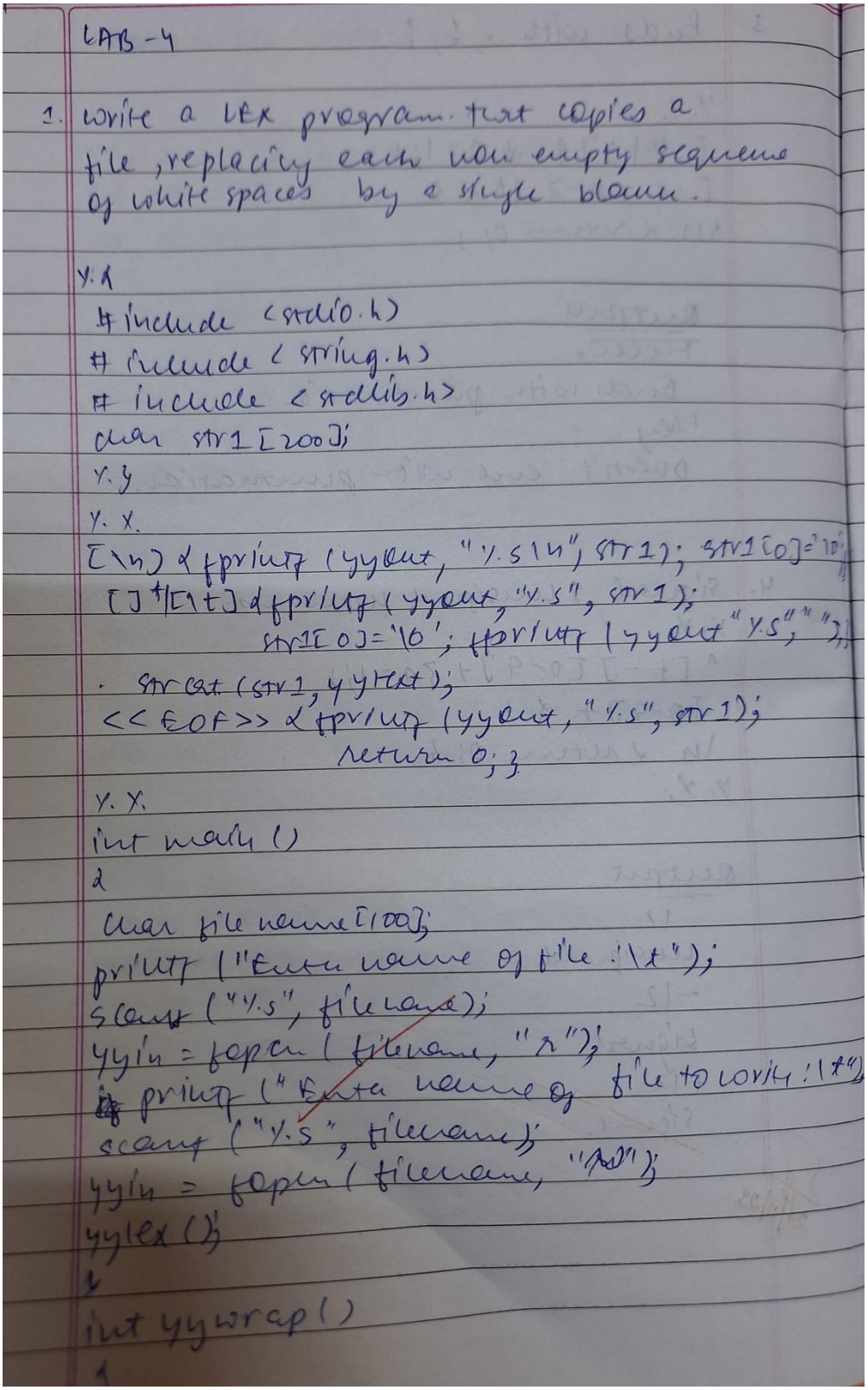
Output

```
Enter a number:
123
Unsigned number!

-123
Signed number!

+123
Signed number!
```

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

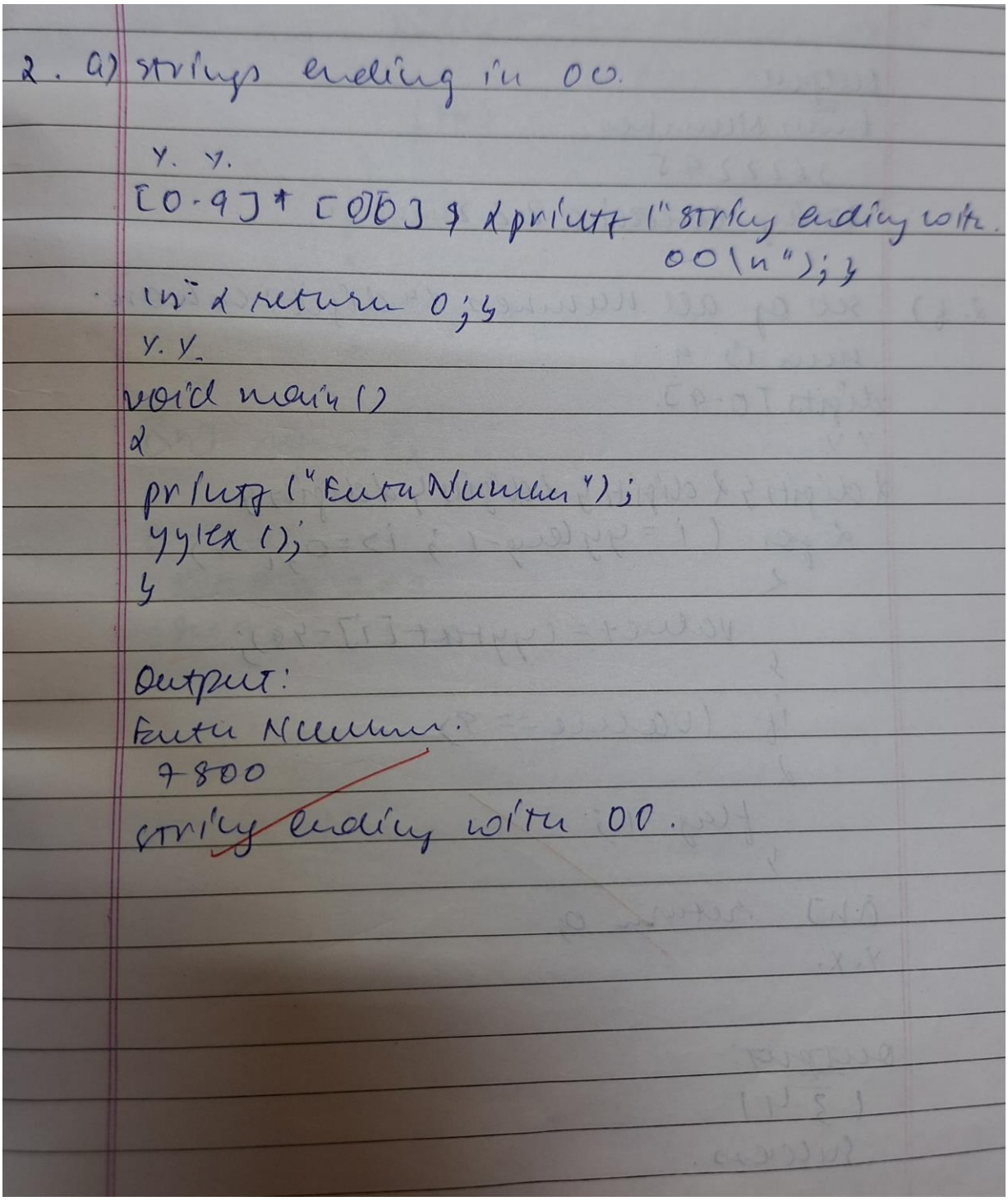

Printed!


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

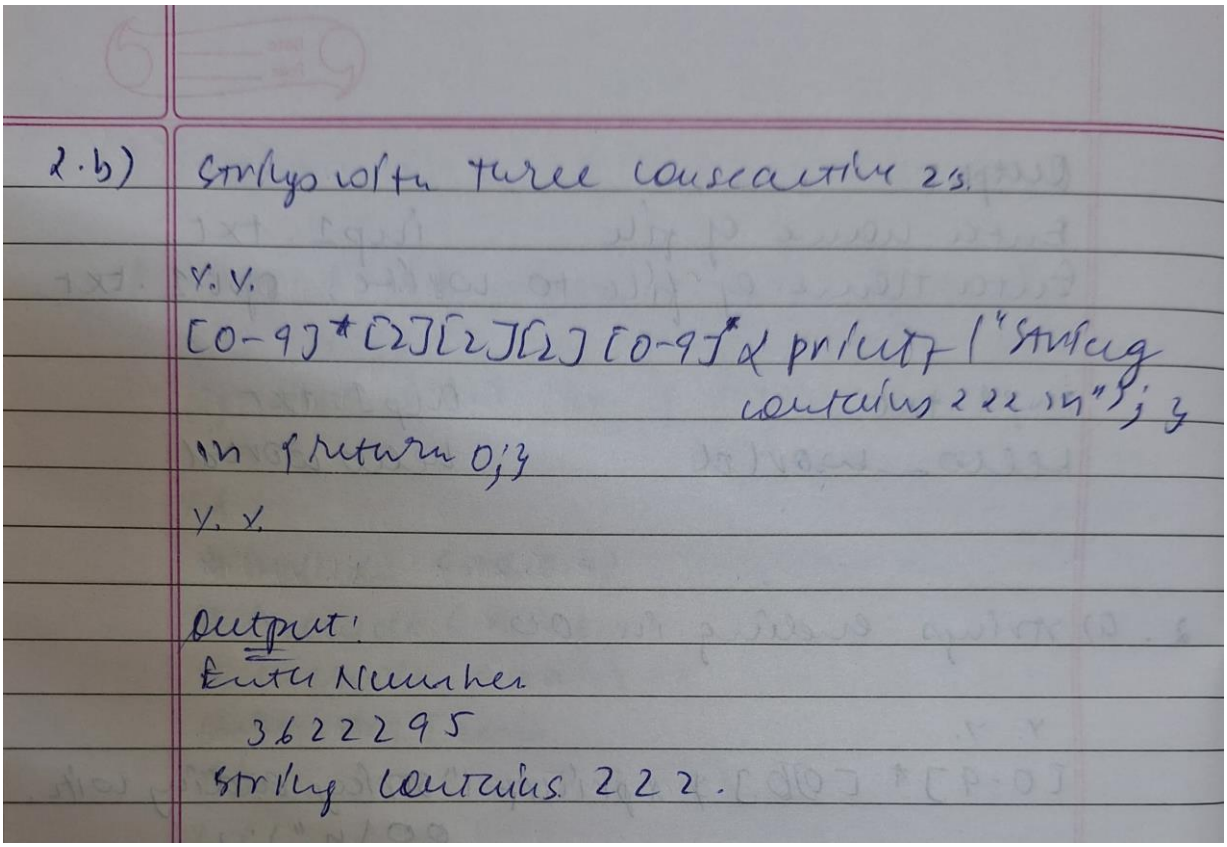


Output


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



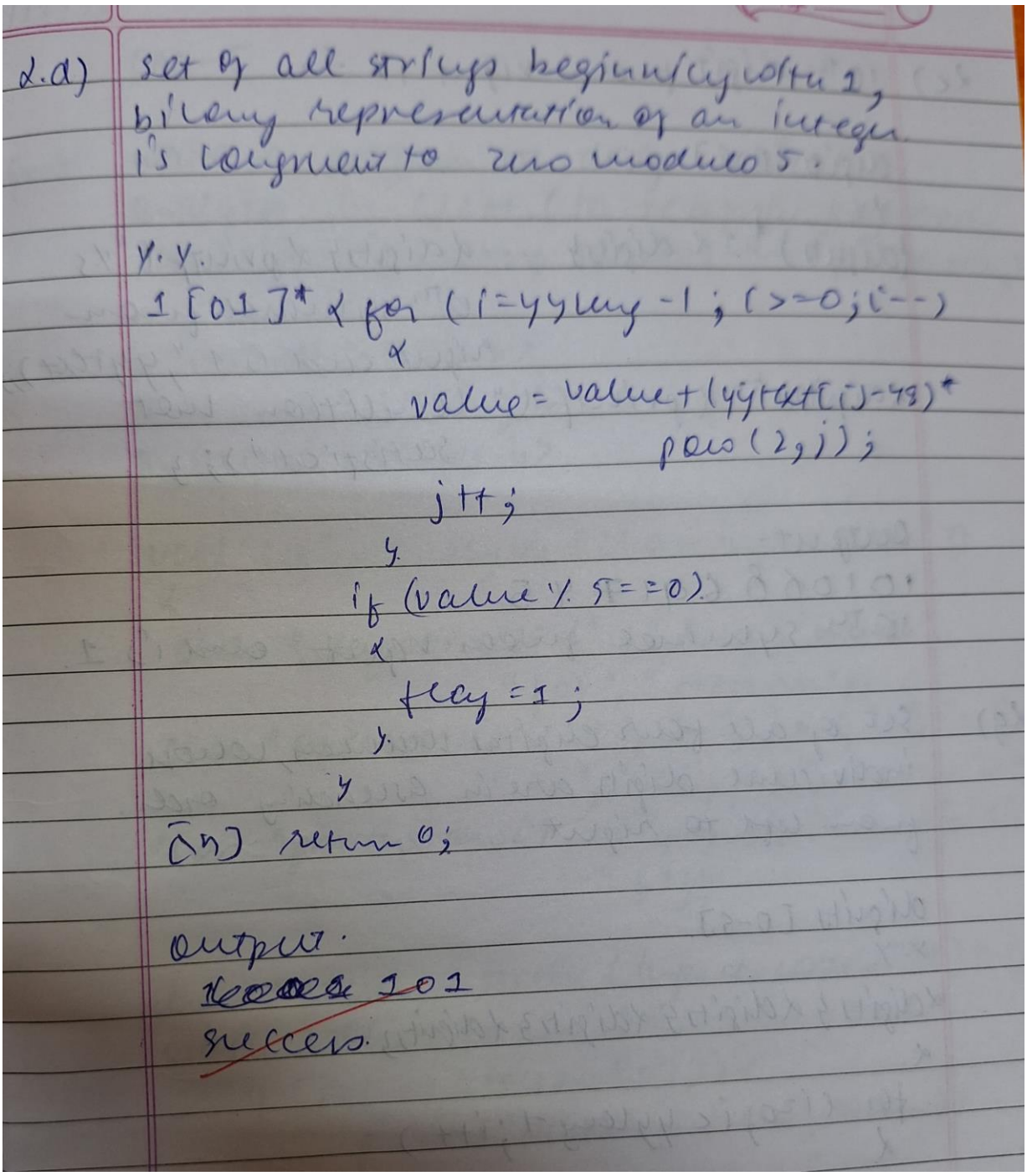
Output

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

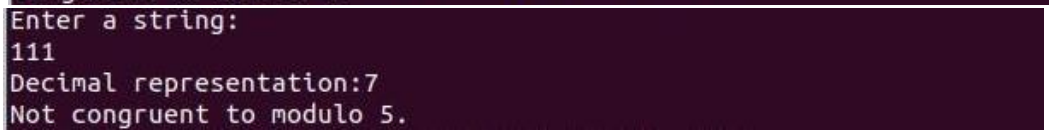
Code



Output


Enter a string:
1010
Decimal representation:10
Congruent to modulo 5.

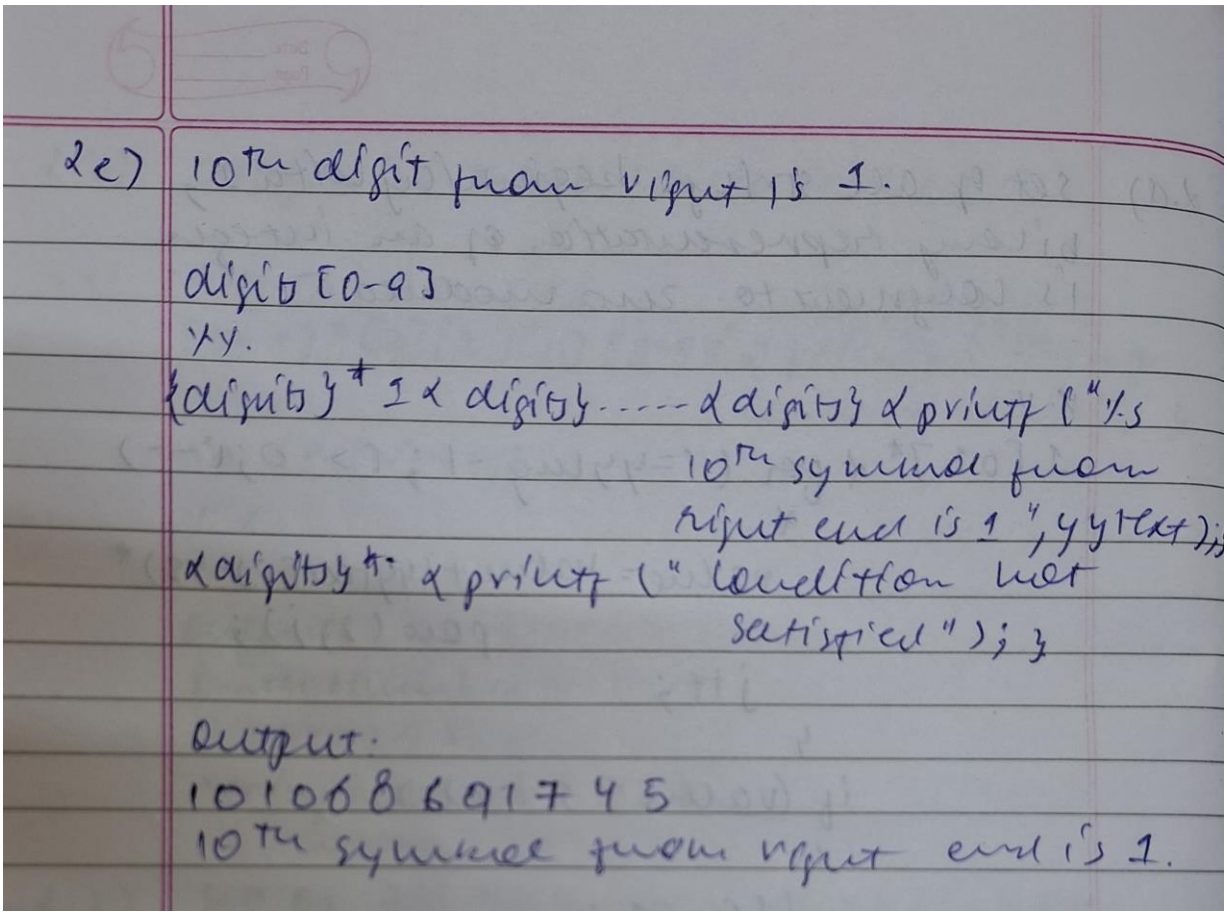

Enter a string:
101
Decimal representation:5
Congruent to modulo 5.


Enter a string:
111
Decimal representation:7
Not congruent to modulo 5.


Enter a string:
123
Not a binary number.

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

Code

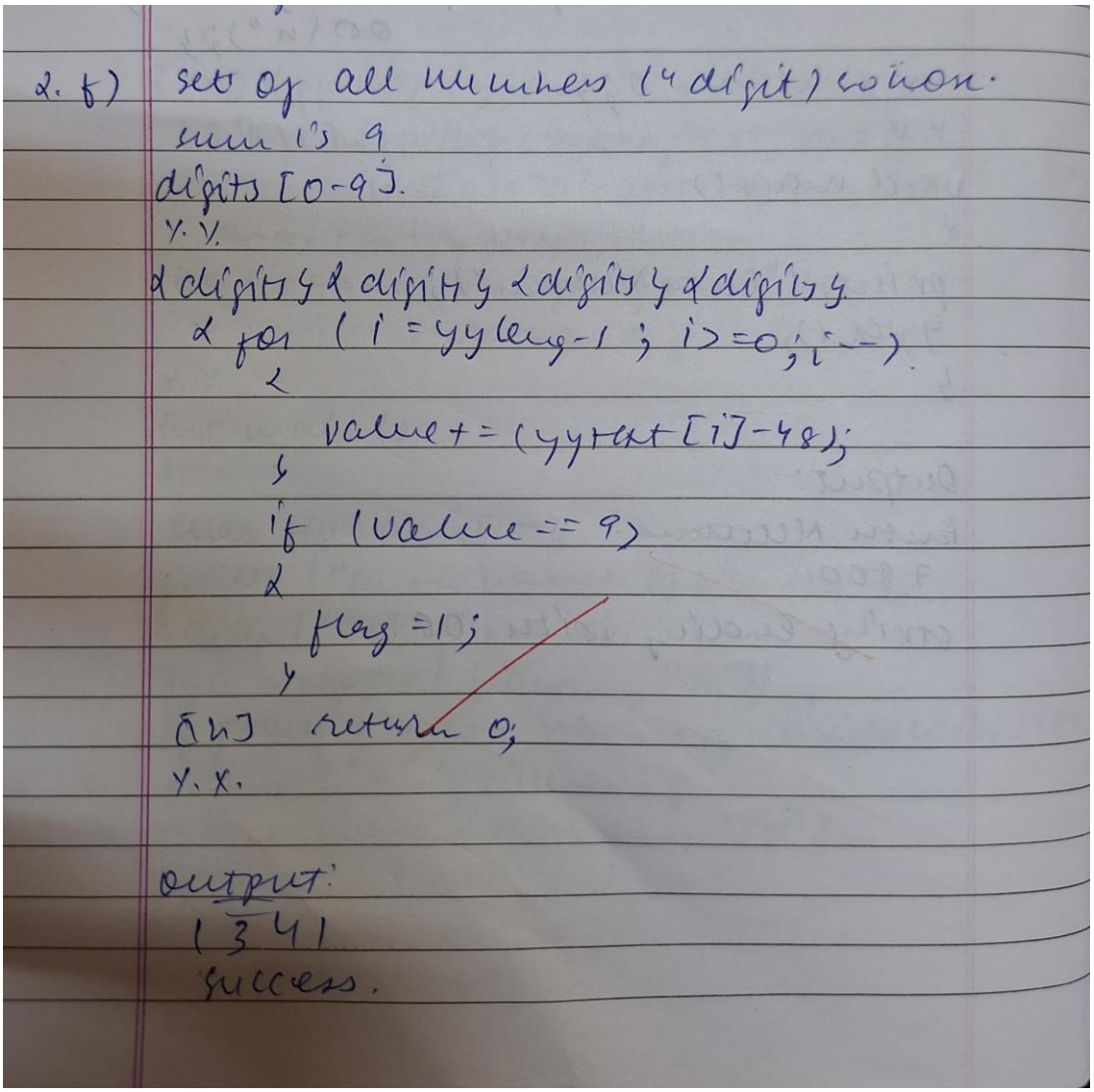


Output

```
Enter a string:
23123456123
10th symbol from right is not 1.
Enter a string:
11234345236
10th symbol from right is 1.
```

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

Code



2. f) set of all numbers (4 digit) whose sum is 9
digits [0-9].
Y. Y.
{ digits } { digits } { digits } { digits }
{ for (i = strlen-1 ; i >= 0 ; i--)
{
value += (str[i] - 48);
}
if (value == 9)
{
flag = 1;
}
[N] return 0;
Y. X.

output:
1341
success.

Output

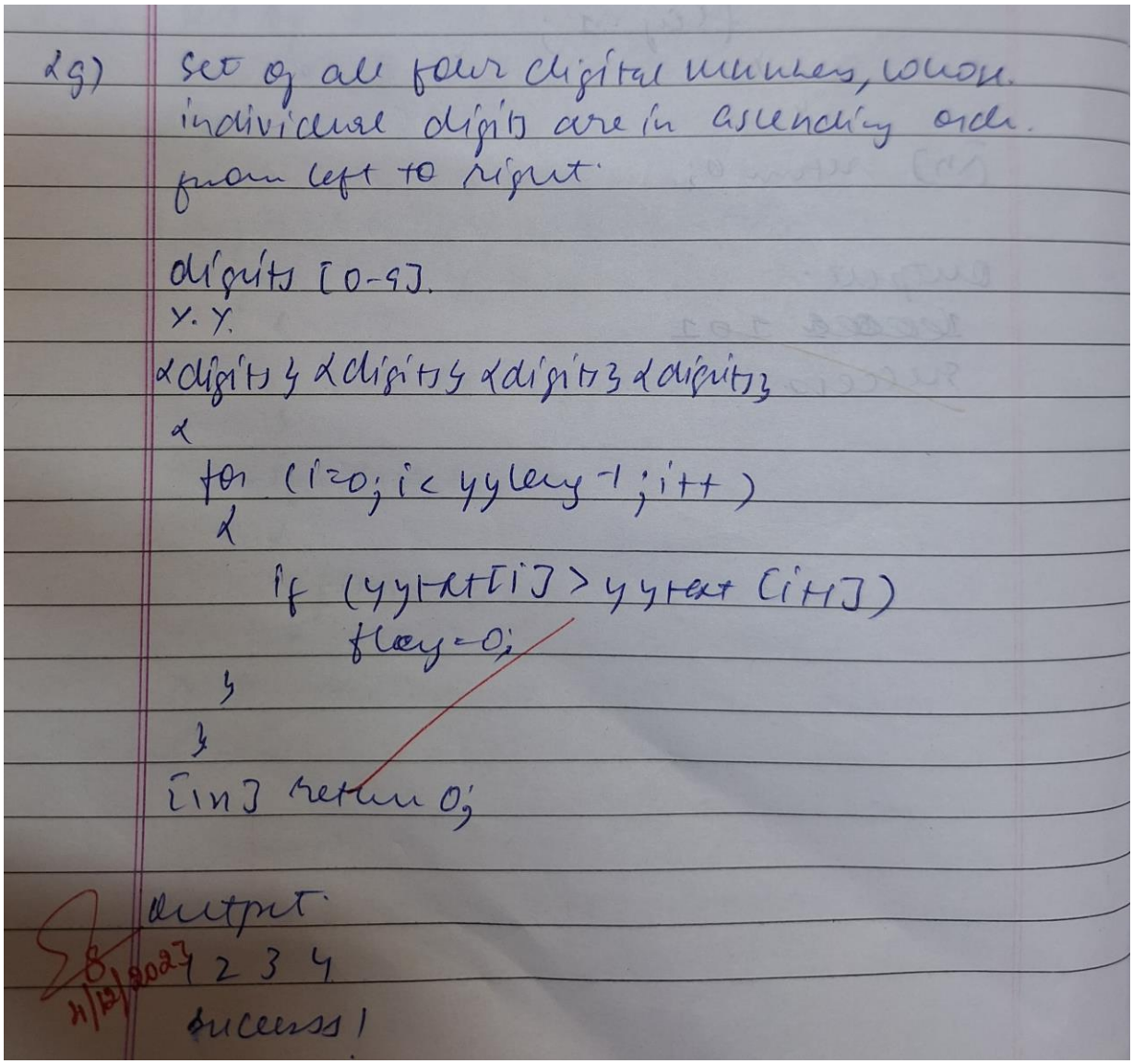
```
Enter a string:
6300
The sum of digits is 9.
```

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

```
Enter a string:
2340
The sum of digits is 9.
```

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

Code



Output

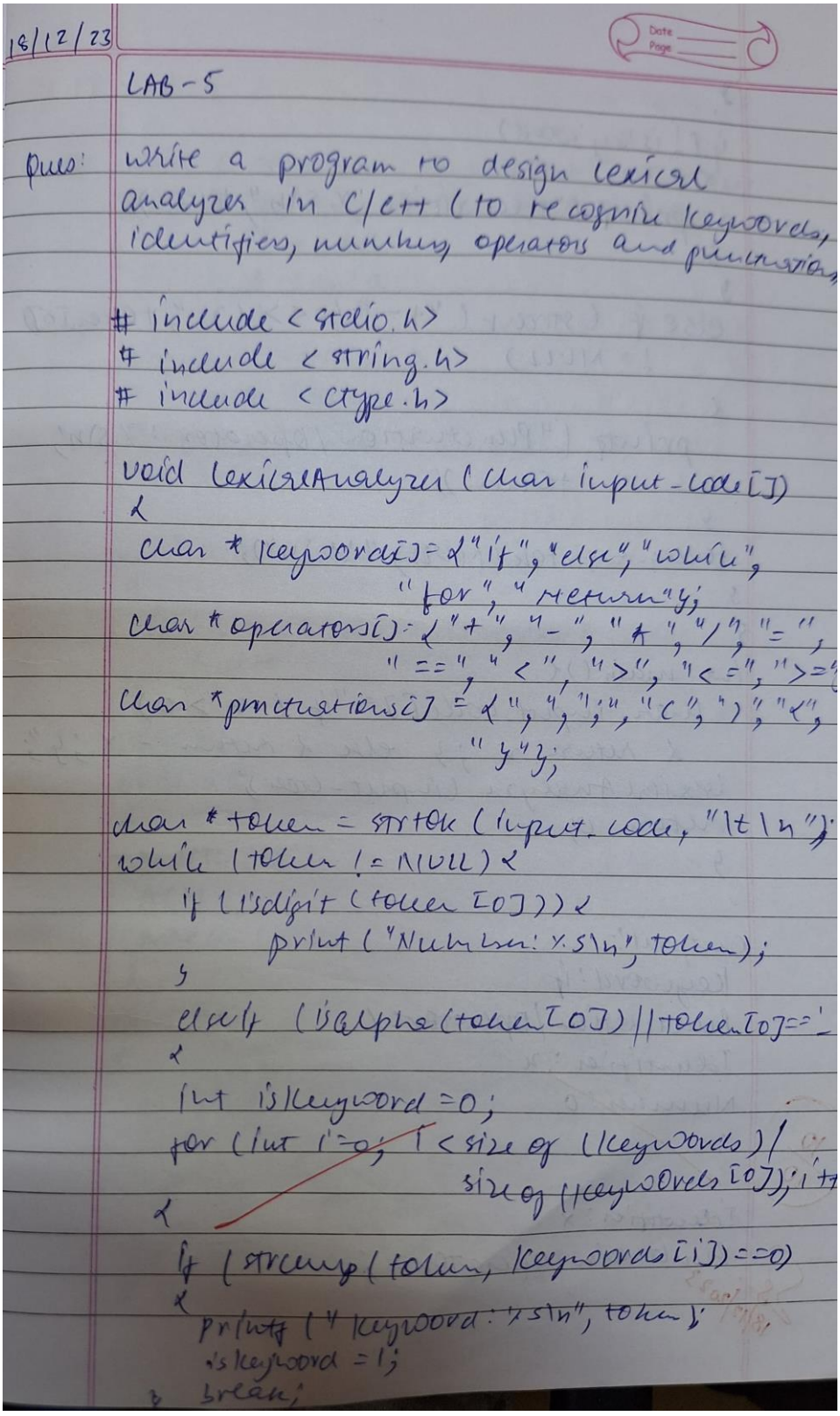
```
Enter a string:
1235
The digits are in ascending order.
```

```
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



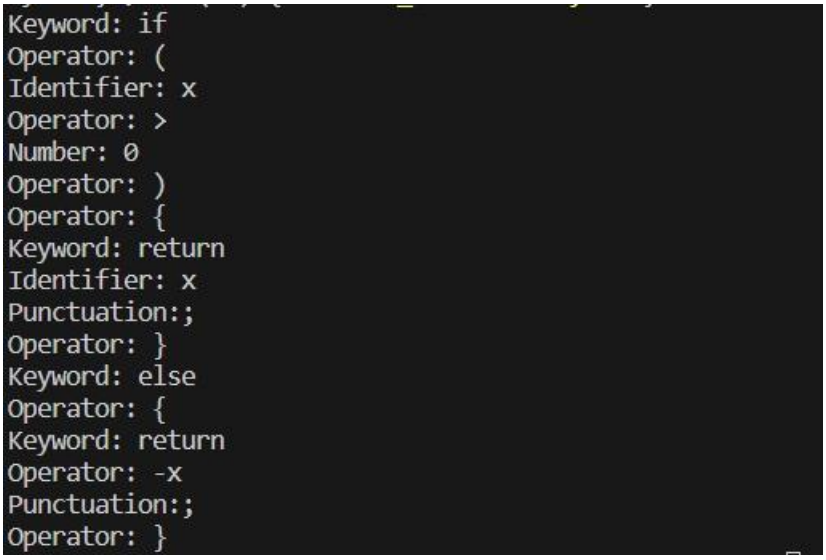
```
16/12/23

LAB-5

Aim: write a program to design lexical
analyzer in C/C++ to recognize keywords,
identifiers, numbers, operators and punctuations

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

void lexicalAnalyzer(char input_code[])
{
 char *keywords[] = {"if", "else", "while",
                     "for", "return"};
 char *operators[] = {"+", "-", "*", "/", "=",
                      "==", "<", ">", "<=", ">="
 char *punctuations[] = {",", ";", ":", "(", ")", "{",
                         "}"};

char *token = strtok(input_code, " \t\n");
while (token != NULL) {
    if (isdigit(token[0])) {
        printf("Number: %s\n", token);
    }
    else if (isalpha(token[0]) || token[0] == '_
    {
    int isKeyword = 0;
    for (int i = 0; i < sizeof(keywords) /
                        sizeof(keywords[0]); i++
    {
    if (strcmp(token, keywords[i]) == 0)
    {
      printf("Keyword: %s\n", token);
      isKeyword = 1;
    } break;
```

```
}
if (!isKeyword)
{
    printf("Identifier : %s\n", token);
}
}
else if (strchr("+-*/=<>;(),", token[0])
        != NULL)
{
    printf("Punctuation/Operator : %s\n",
            token);
}
token = strtok(NULL, " \t\n");
}
}
int main(){
    char input_code[] = "if (x > 0)
    { return x; } else { return -x; }";
lexicalAnalyzer(input_code);
return 0;
}
```

Output:-

Keyword : if
Punctuation/Operator : (
Identifier : x
Number : 0
:
:
Identifier : x
Punctuation/Operator : ;

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: }

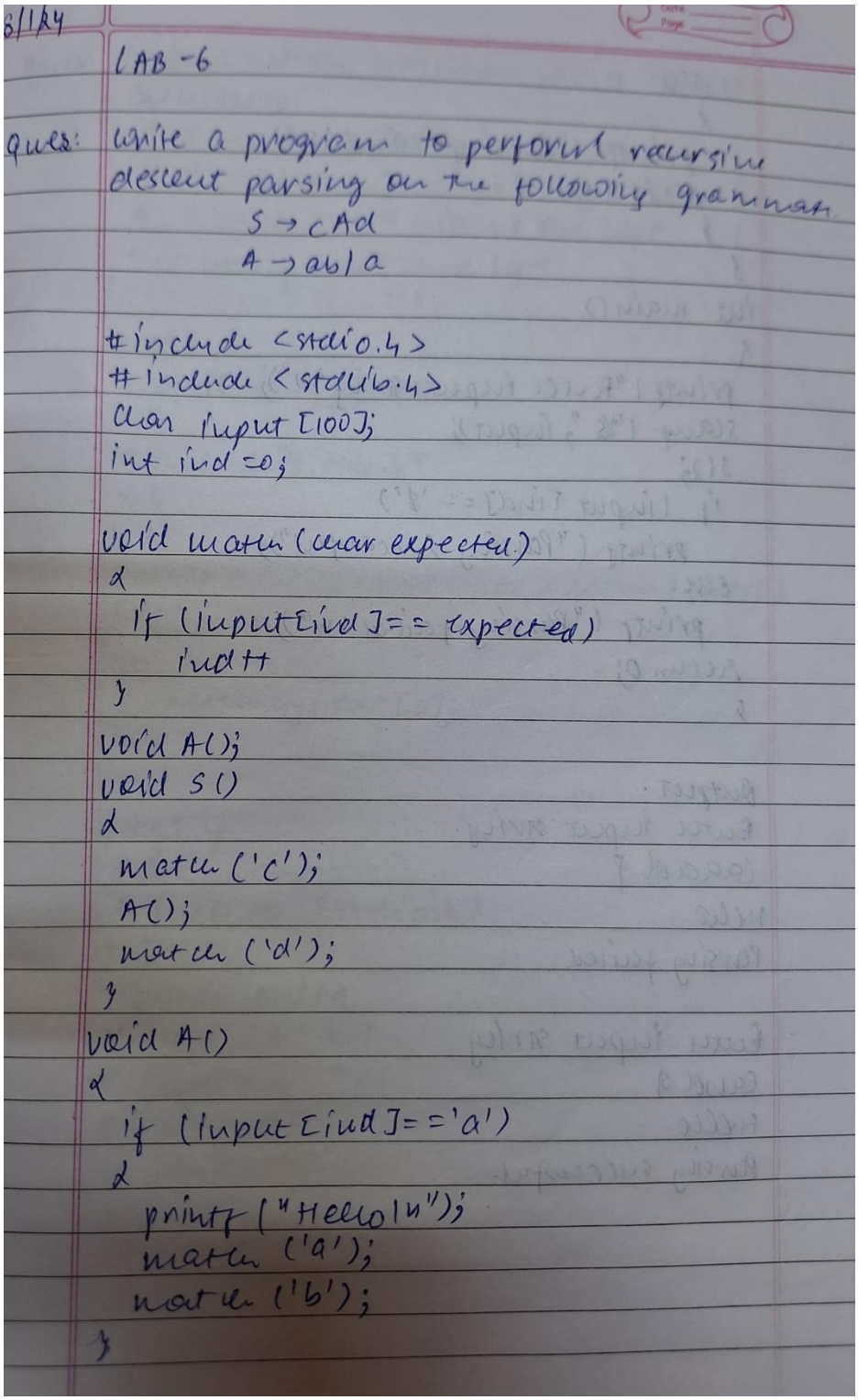
Lab 6

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

S->cAd

A->ab | a

Code



```
6/12/24
LAB-6

Ques: Write a program to perform recursive
descent parsing on the following grammar.
        S → cAd
        A → ab | a

#include <stdio.h>
#include <stdlib.h>
char input[100];
int ind=0;

void match(char expected)
{
  if (input[ind]== expected)
    ind++
}
void A();
void S()
{
 match('c');
 A();
 match('d');
}
void A()
{
 if (input[ind]=='a')
 {
   printf("Hello\n");
   match('a');
   match('b');
}
```

```
    else.
    {
     printf ("Parsing failed.\n")
     exit (1);
    }
}
int main()
{.
 printf ("Enter input string\n");
 scanf ("%s", input);
 S();
 if (input[ind] == '$')
    printf ("Parsing successful");
 else.
    printf ("Parsing failed\n");
 return 0;
}
```

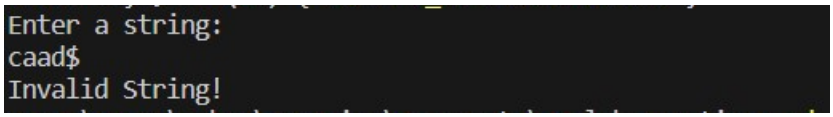
Output.

Enter input string.
caaad$
Hello
Parsing failed.

Enter input string
caad$
Hello
Parsing successful.

Output


Enter a string:
cad$
Valid string!

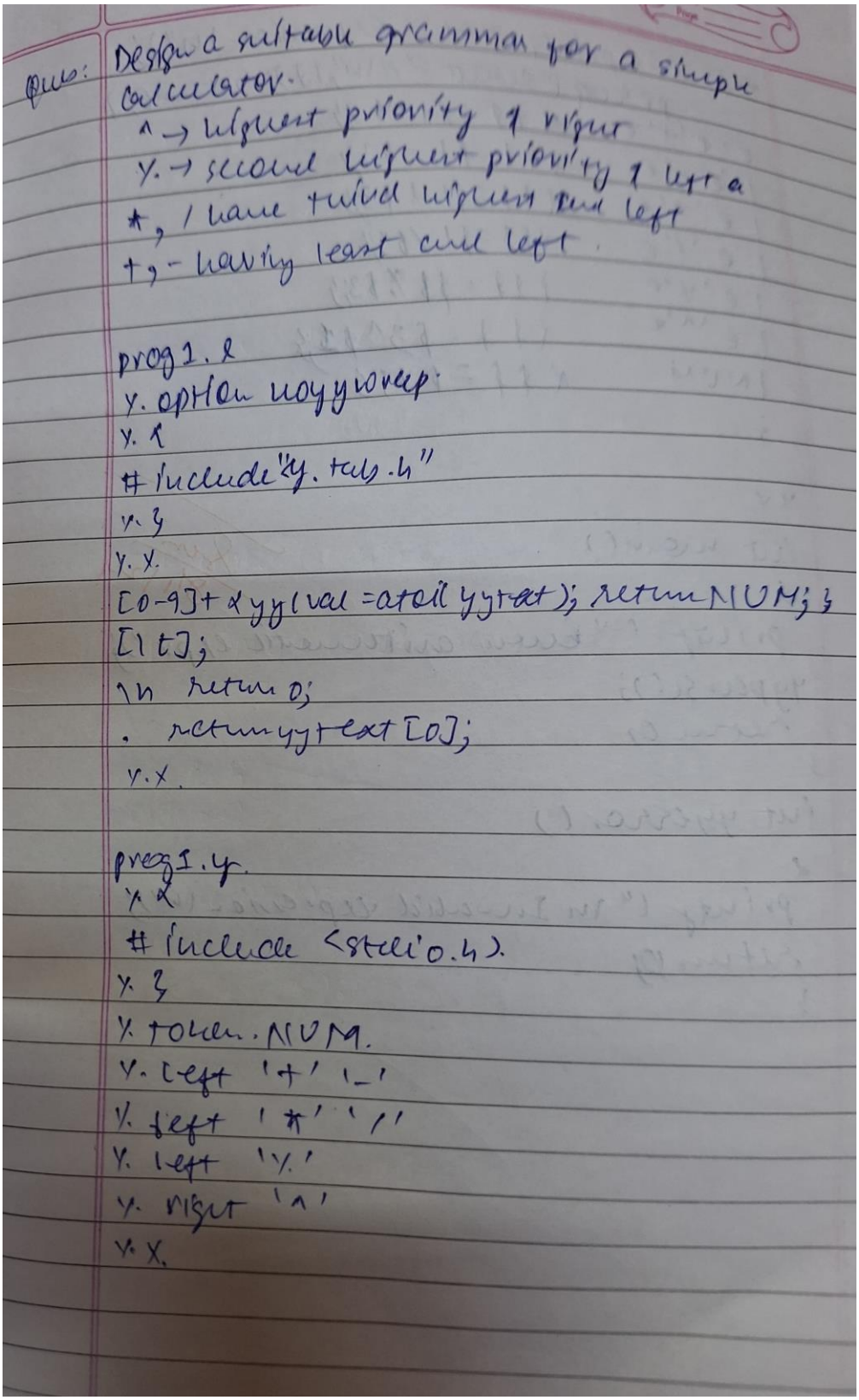

Enter a string:
caad$
Invalid String!

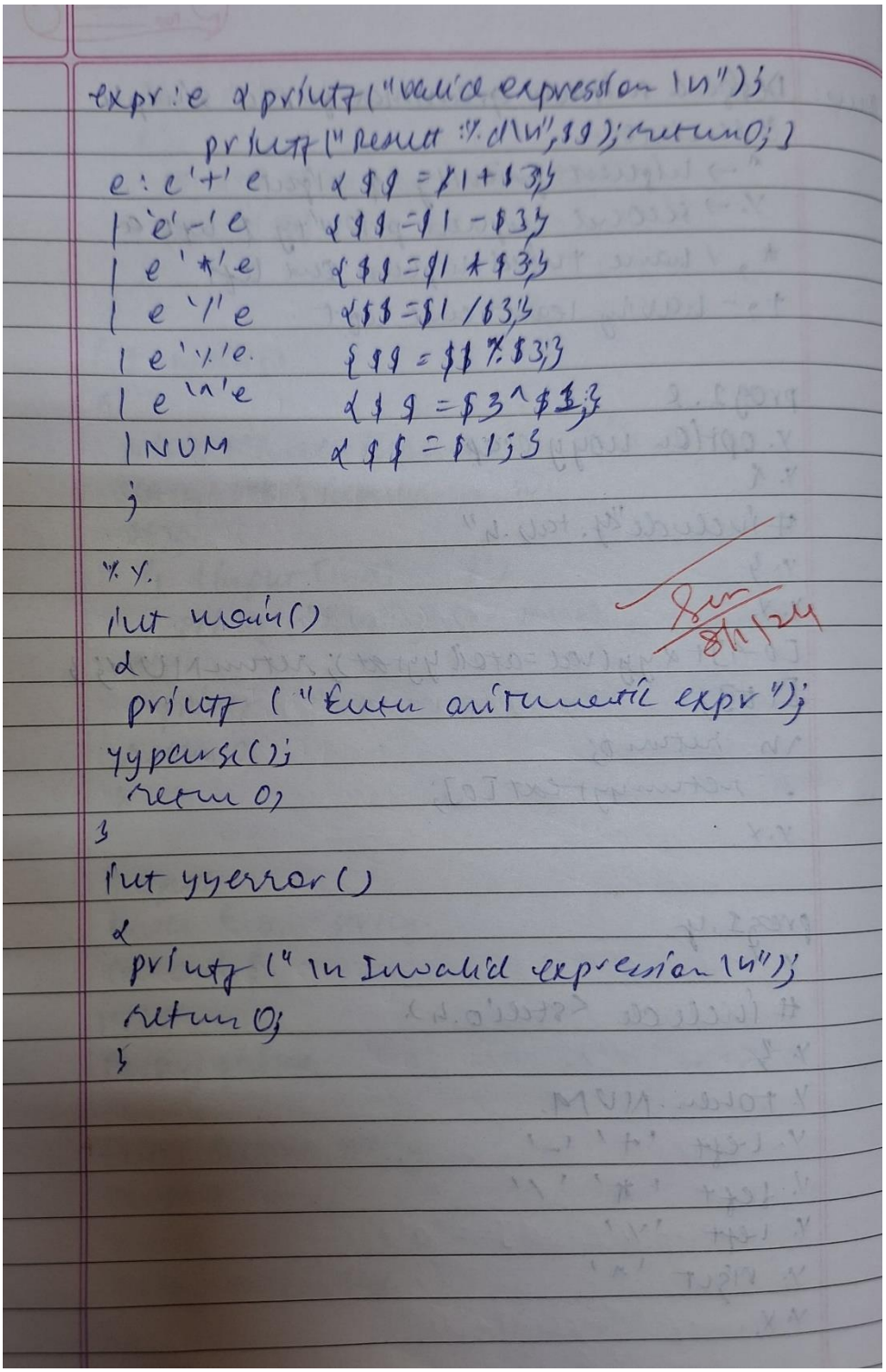

Enter a string:
cabd$
Valid string!

Lab 7

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

Code





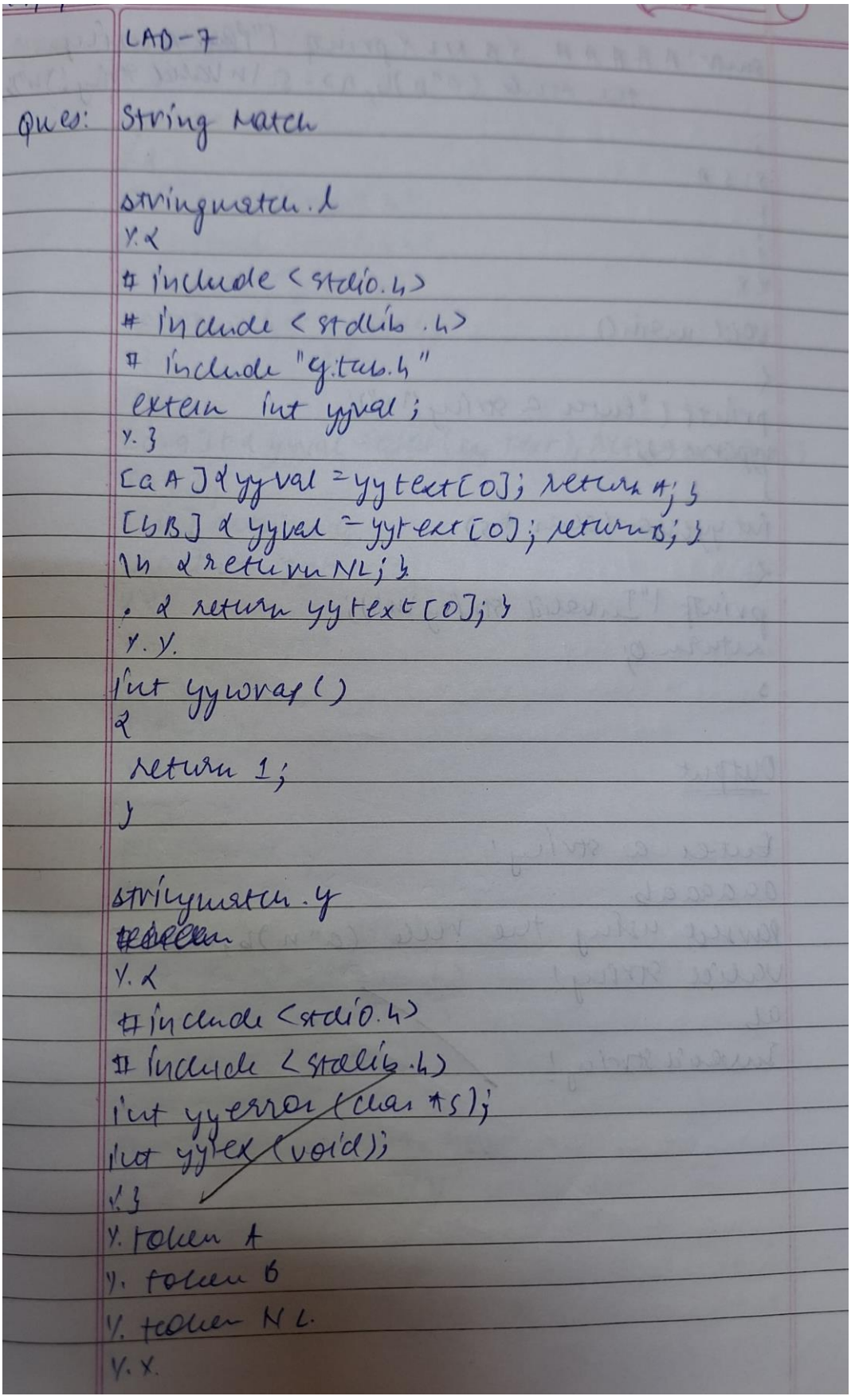
Output

```
Enter an arithmetic expression:
2++3-
Invalid expression!
```

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

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

Code



LAB-7

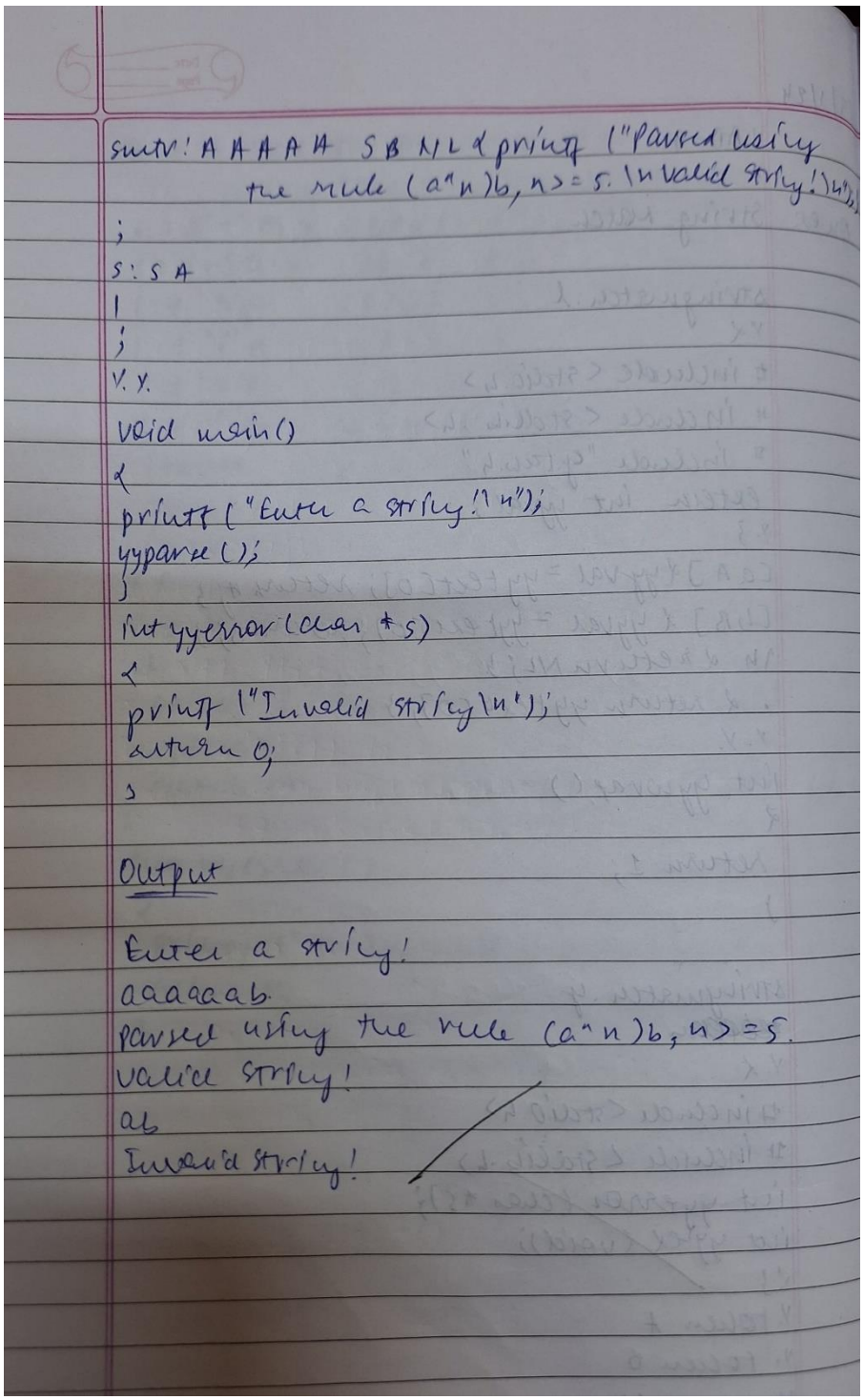
Ques: String Match

stringmatch.l

```
%{
#include <stdio.h>
#include <stdlib.h>
#include "y.tab.h"
extern int yyval;
%}
[aA] { yyval = yytext[0]; return A; }
[bB] { yyval = yytext[0]; return B; }
\n { return NL; }
. { return yytext[0]; }
%%
int yywrap()
{
return 1;
}
```

stringmatch.y

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

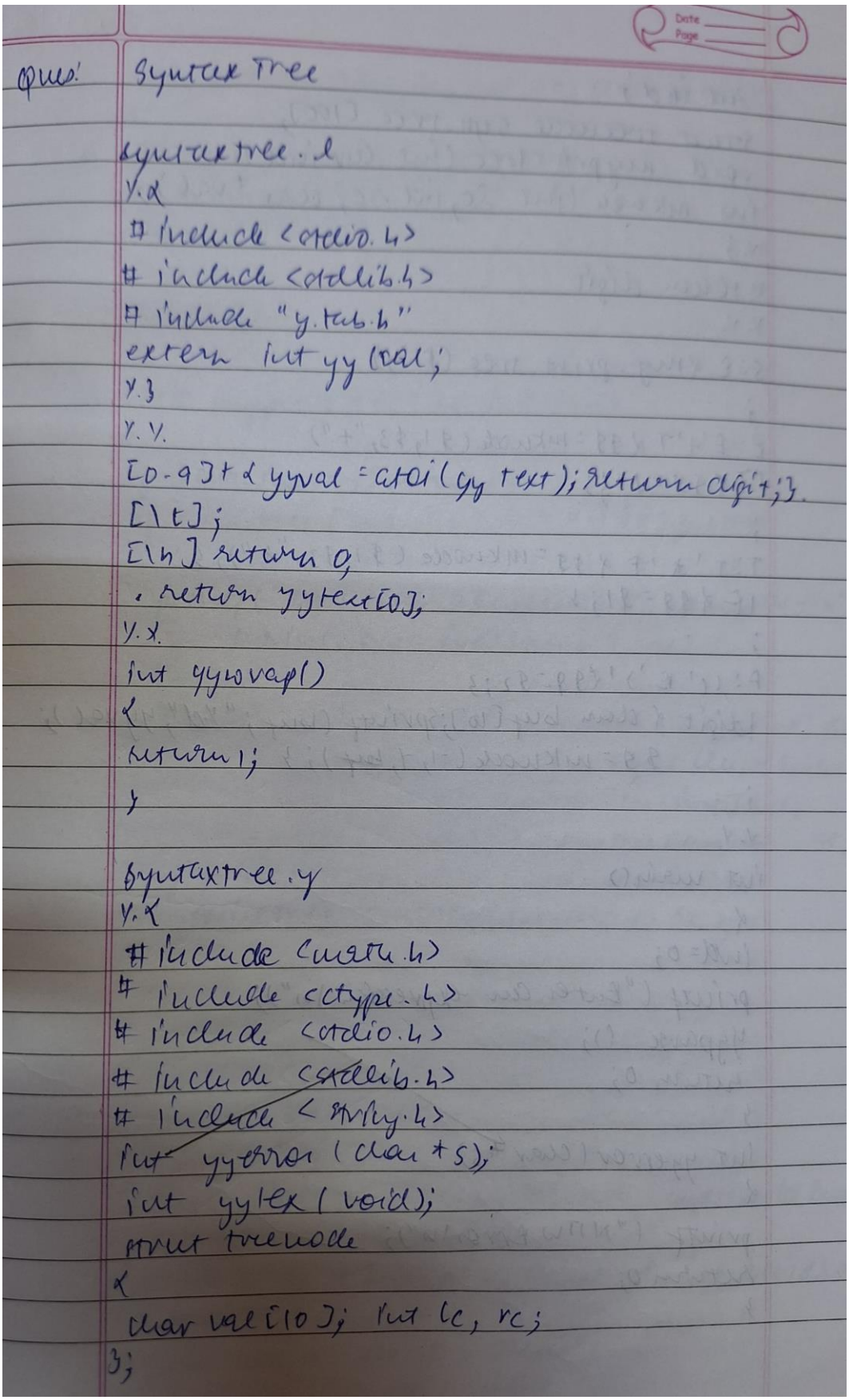


Output

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

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

Code



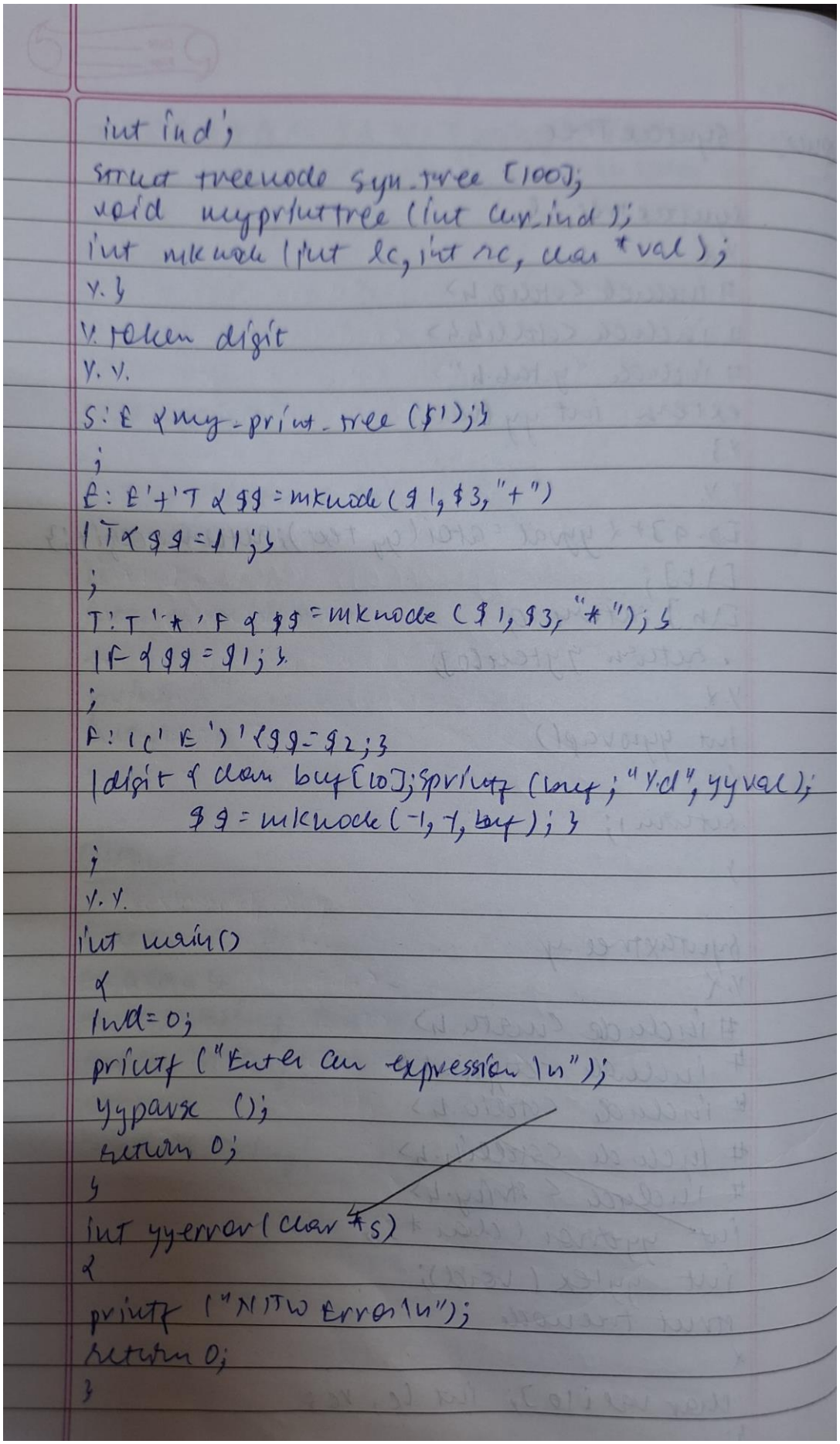
Ques: Syntax Tree

syntaxtree.l

```
%{
#include <stdio.h>
#include <stdlib.h>
#include "y.tab.h"
extern int yylval;
%}
%%
[0-9]+ { yylval = atoi(yytext); return digit; }
[\t];
[\n] return 0;
. return yytext[0];
%%
int yywrap()
{
return 1;
}
```

syntaxtree.y

```
%{
#include <math.h>
#include <ctype.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
int yyerror(char *s);
int yylex(void);
struct treenode
{
char val[10]; int lc, rc;
};
```



```
int ind;
struct treenode syn_tree[100];
void myprinttree(int cur_ind);
int mknode(int lc, int rc, char *val);
%}
%token digit
%%
S: E { my_print_tree($1); }
;
E: E '+' T { $$ = mknode($1, $3, "+") }
| T { $$ = $1; }
;
T: T '*' F { $$ = mknode($1, $3, "*"); }
| F { $$ = $1; }
;
F: '(' E ')' { $$ = $2; }
| digit { char buf[10]; sprintf(buf, "%d", yyval);
          $$ = mknode(-1, -1, buf); }
;
%%
int main()
{
ind=0;
printf("Enter an expression \n");
yyparse();
return 0;
}
int yyerror(char *s)
{
printf("\nITS Error\n");
return 0;
}
```

```
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 myprinttree (int cur_ind)
{
if (cur_ind == -1) return;
if (syn_tree [cur_ind].lc == -1 && syn_tree
    [cur_ind].rc == -1)
  printf("Digit Node -> Index : %d, Value : %s\n",
      cur_ind, syntree[cur_ind].val);
  else
  printf("Operator Node -> Index : %d, Value : %s,
      Left child Index : %d, Right child Index :
      %d\n", cur_ind, syntree[cur_ind].val,
      syntree[cur_ind].lc, syntree[cur_ind].rc);
my printtree (syntree[cur_ind].lc);
myprinttree (syntree[cur_ind].rc);
}
```

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
Index : 1 value : 3.

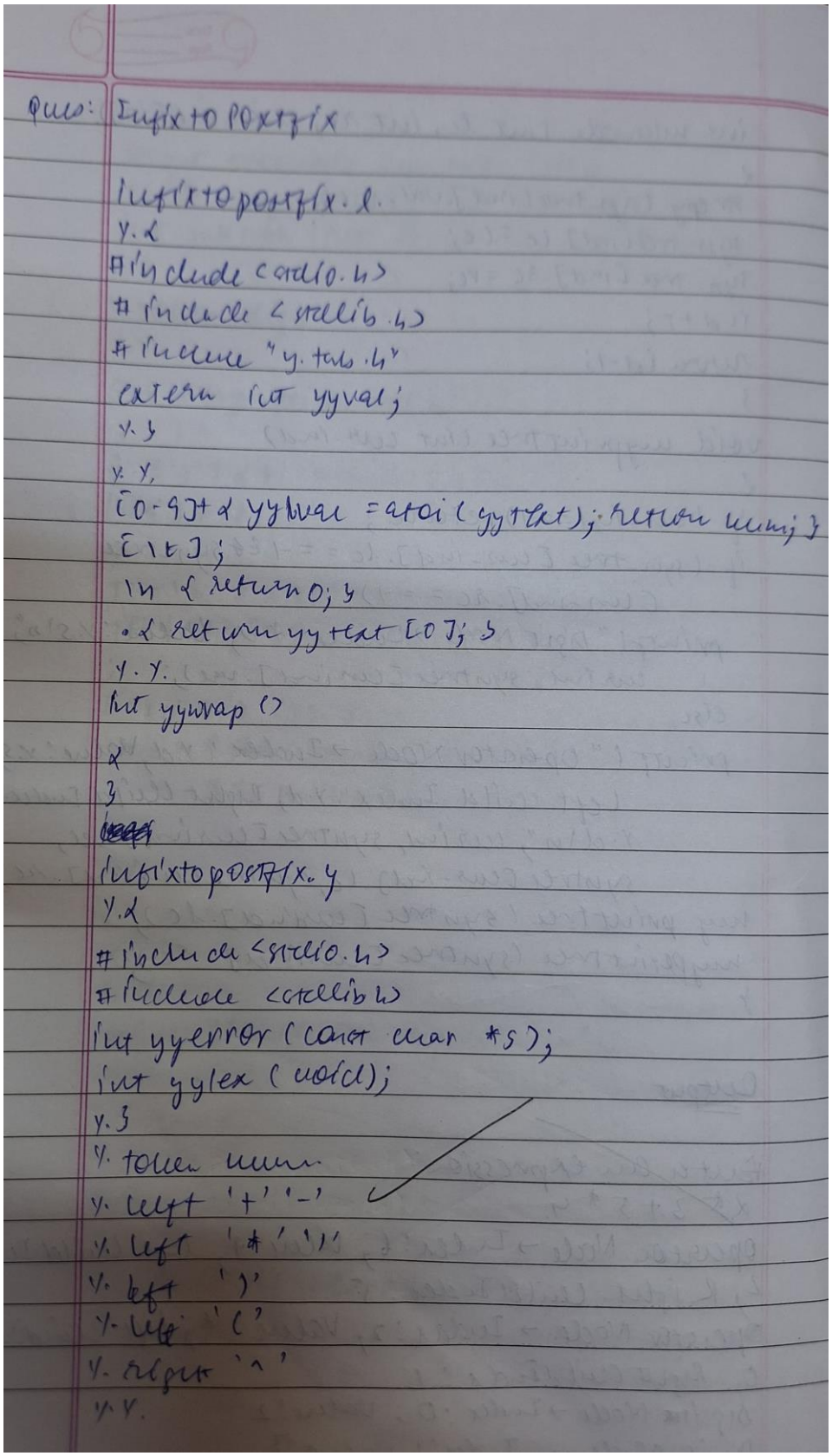
Output

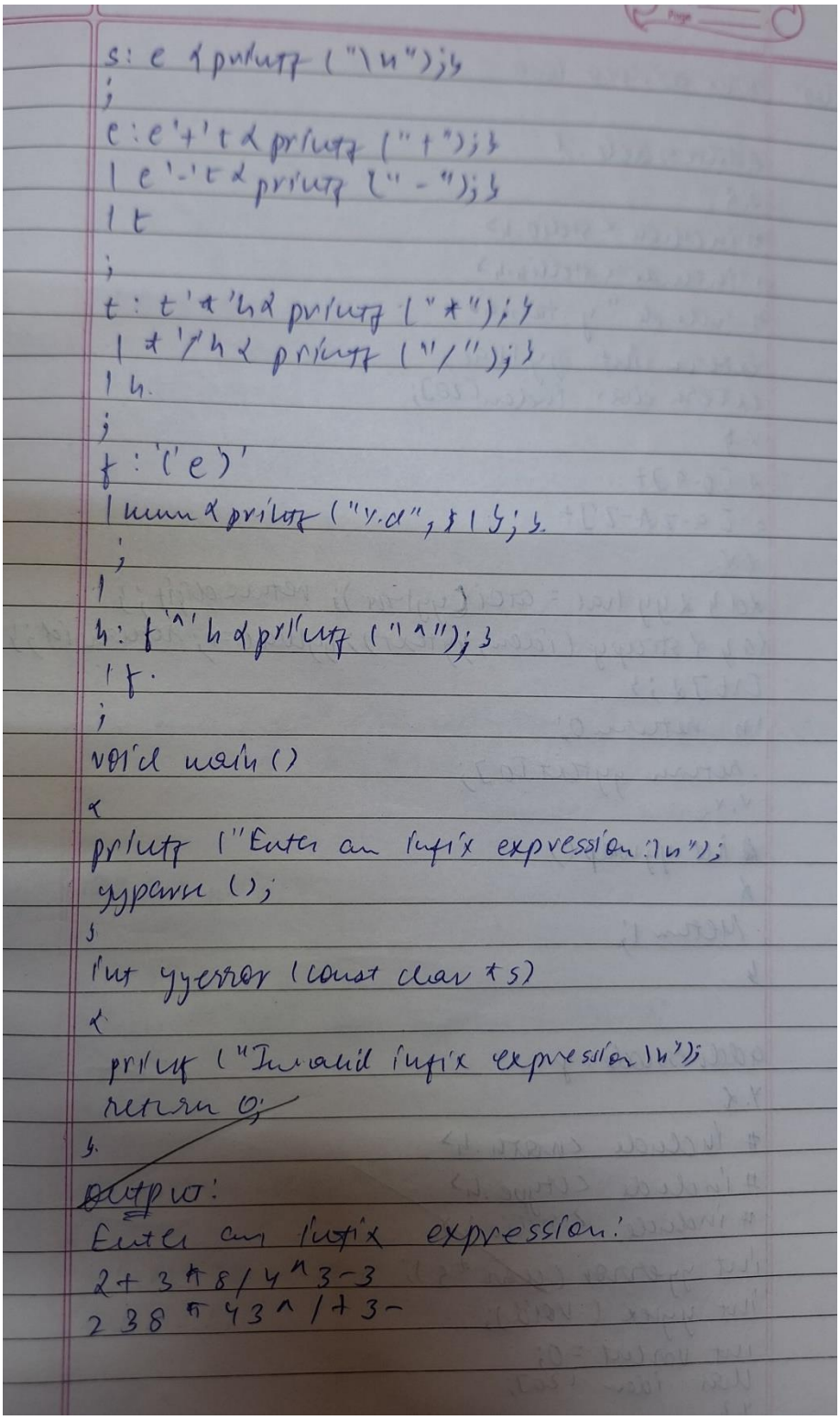
```
Enter an expression:
2*3+5*4
Operator Node -> Index : 6, Value : +, Left Child Index : 2,Right Child Index : 5
Operator Node -> Index : 2, Value : *, Left Child Index : 0,Right Child Index : 1
Digit Node -> Index : 0, Value : 2
Digit Node -> Index : 1, Value : 3
Operator Node -> Index : 5, Value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





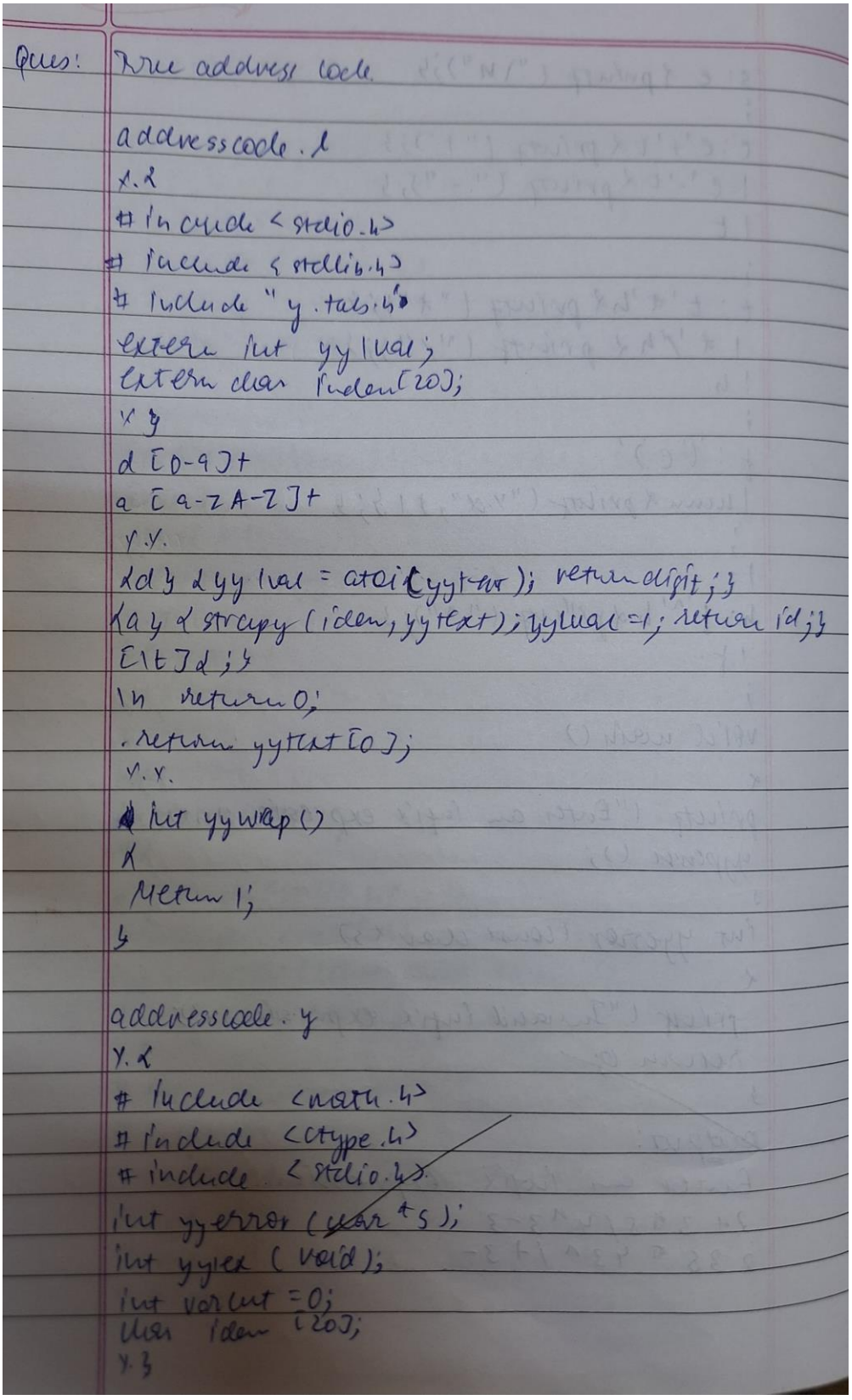
Output

```
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



Ques: Three address code

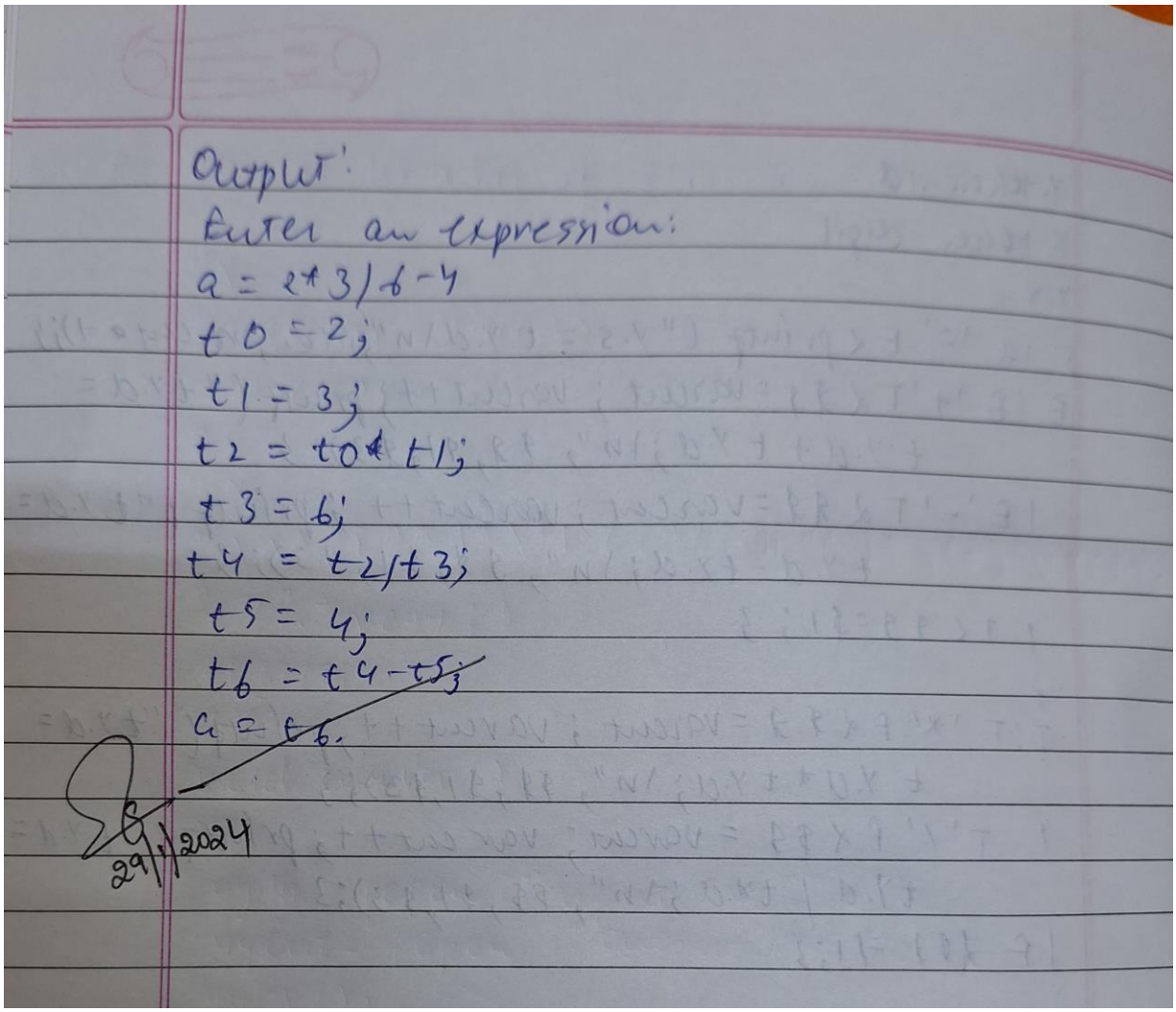
addresscode.l

```
%.%
#include <stdio.h>
#include <stdlib.h>
#include "y.tab.h"
extern int yylval;
extern char iden[20];
%%
[0-9]+
[a-zA-Z]+
%%
[0-9]+ {yylval = atoi(yytext); return digit;}
[a-zA-Z]+ {strcpy(iden, yytext); yylval=1; return id;}
[\t]+;
\n return 0;
. return yytext[0];
%%
int yywrap()
{
return 1;
}
```

addresscode.y

```
%%
#include <math.h>
#include <ctype.h>
#include <stdio.h>
int yyerror(char *s);
int yylex(void);
int varcnt=0;
char iden[20];
%%
```

```
%token id
%token digit
%%
S: id '=' E {printf("%s = t%d\n", id.en, varcnt-1);}
E: E '+' T {$$ = varcnt; varcnt++; printf("t%d = t%d + t%d;\n", $$, $1, $3);}
 | E '-' T {$$ = varcnt; varcnt++; printf("t%d = t%d - t%d;\n", $$, $1, $3);}
 | T {$$ = $1;}
 ;
T: T '*' F {$$ = varcnt; varcnt++; printf("t%d = t%d * t%d;\n", $$, $1, $3);}
 | T '/' F {$$ = varcnt; varcnt++; printf("t%d = t%d / t%d;\n", $$, $1, $3);}
 | F {$$ = $1;}
 ;
F: P '^' F {$$ = var_cnt; var_cnt++; printf("t%d = %d ^ %d;\n", $$, $1, $3);}
 | P {$$ = $1;}
 ;
%%
int main()
{
varcnt = 0;
printf("Enter an expression:\n");
yyparse();
return 0;
}
int yyerror(char *s)
{
printf("Invalid expression!");
return 0;
}
```



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
```