VISVESVARAYA TECHNOLOGICAL UNIVERSITY

"JnanaSangama", Belgaum -590014, Karnataka.

LAB REPORT

on

COMPILER DESIGN

Submitted by

J Hanuma Kaushik(1BM21CS078)

Under the Guidance of
Prof. Prameetha Pai
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 **J Hanuma Kaushik(1BM21CS078)** , 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. Prameetha Pai	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, J Hanuma Kaushik(1BM21CS078), 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. Prameetha Pai, Assistant Professor, Department of CSE, B. M. S. College of Engineering, Bangalore during the academic semester November-2023-February-2024.

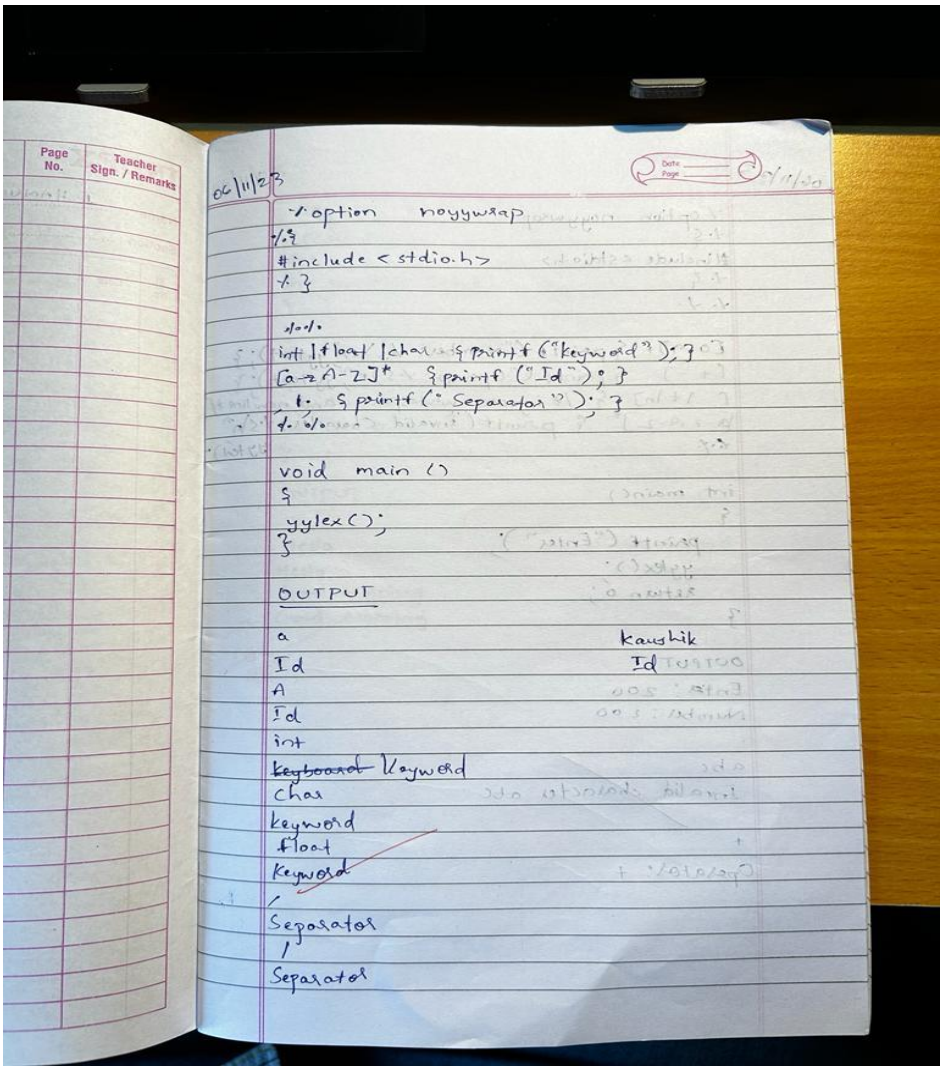
I also declare that to the best of my knowledge and belief, the development reported here is not from part of any other report by any other students.

TABLE OF CONTENTS

Lab 1

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

Code:



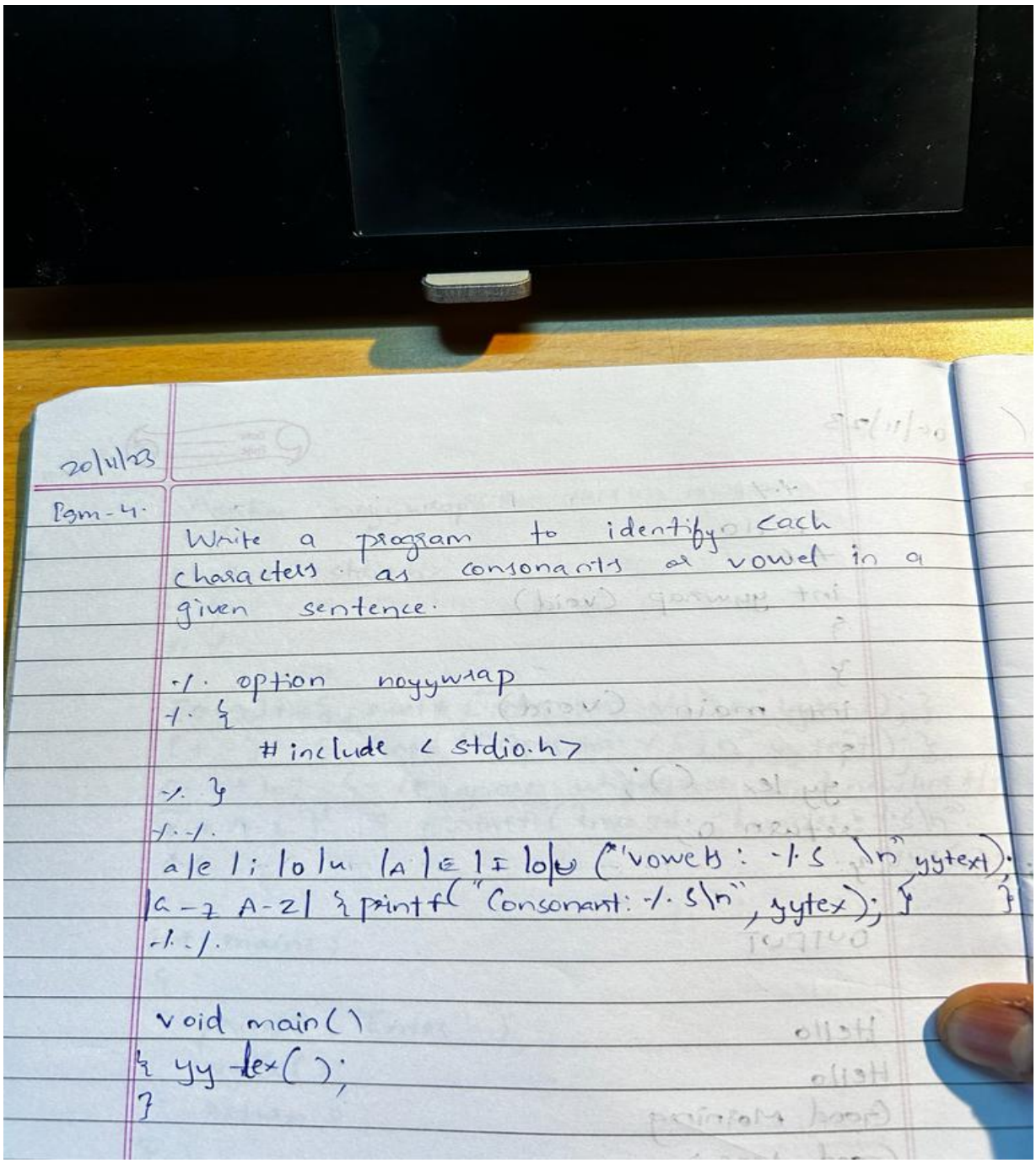
```
%option noyywrap
%{
#include <stdio.h>
%}

%%
int |float |char {printf("keyword");}
[a-zA-Z]* {printf("Id");}
. {printf("Separator");}
%%

void main()
{
yylex();
}

OUTPUT

a                    kaushik
Id                   Id
A
Id
int
keyword
char
keyword
float
Keyword
(
Separator
)
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



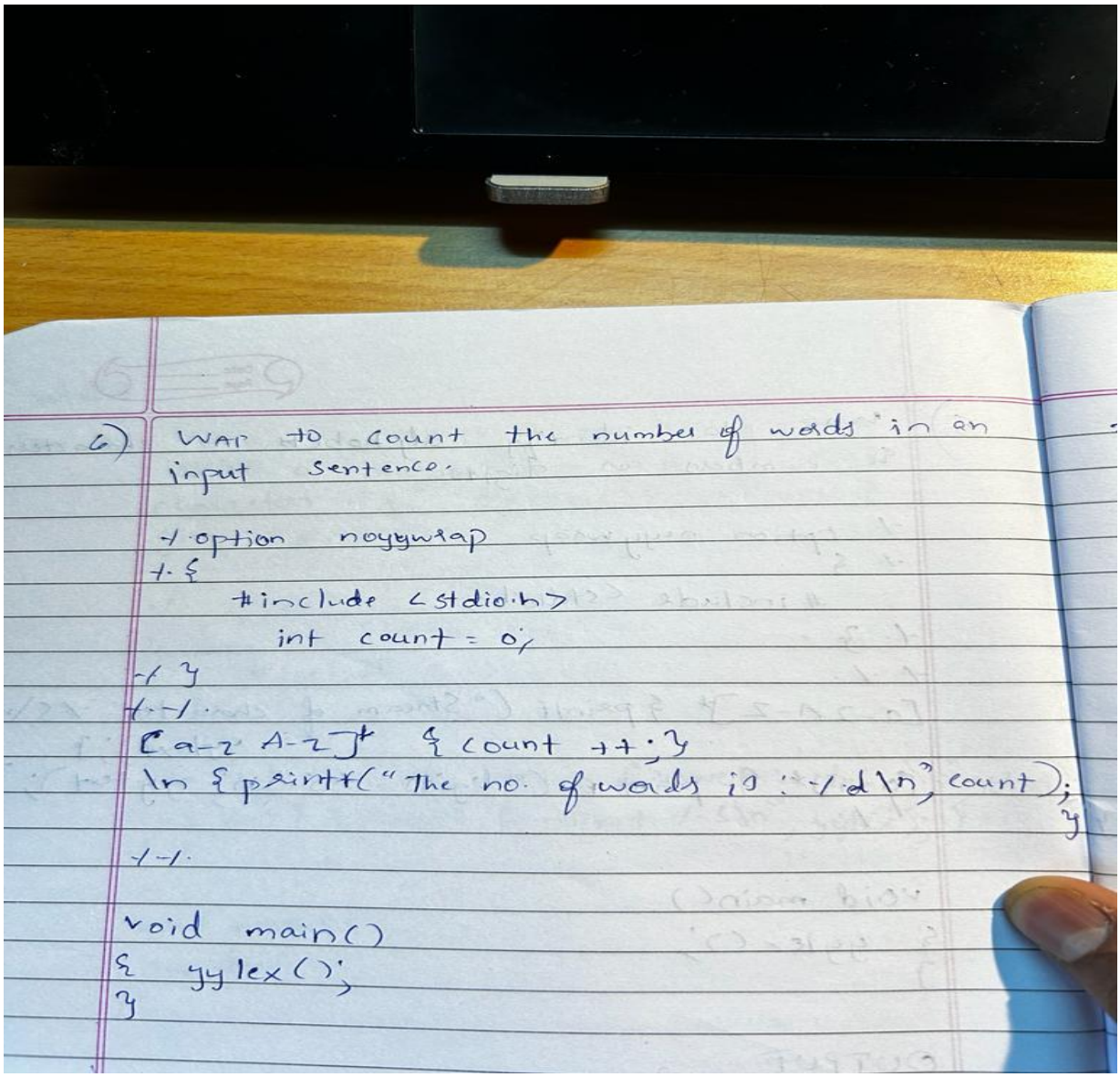
Output

```
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

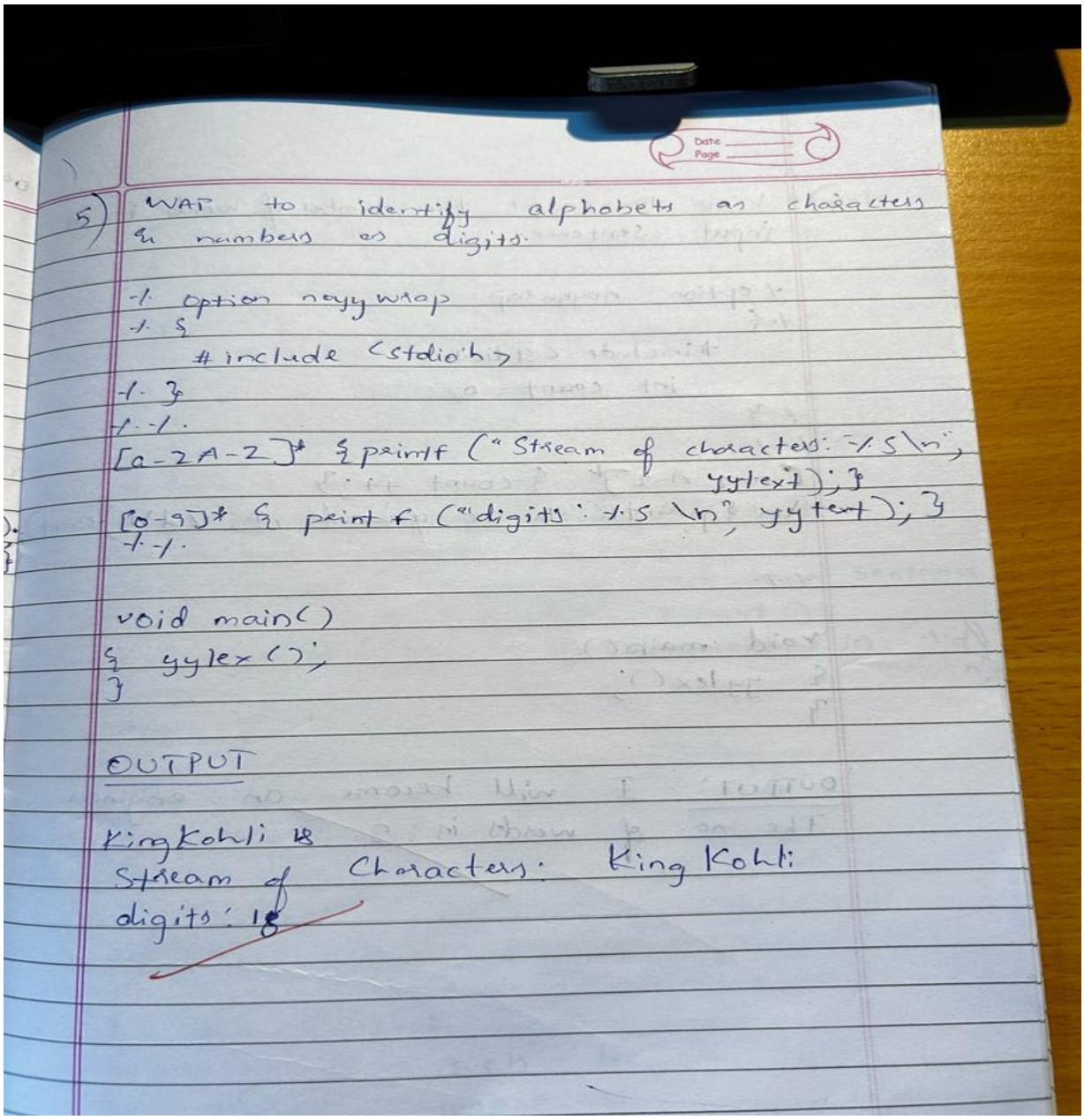


Output

```
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



5) WAP to identify alphabets as characters & numbers as digits.

```
%option noyywrap
%{
    #include<stdio.h>
%}
%%
[a-zA-Z]* {printf("Stream of characters: %s\n", yytext);}
[0-9]* {printf("digits: %s\n", yytext);}
%%

void main()
{ yylex();
}
```

OUTPUT

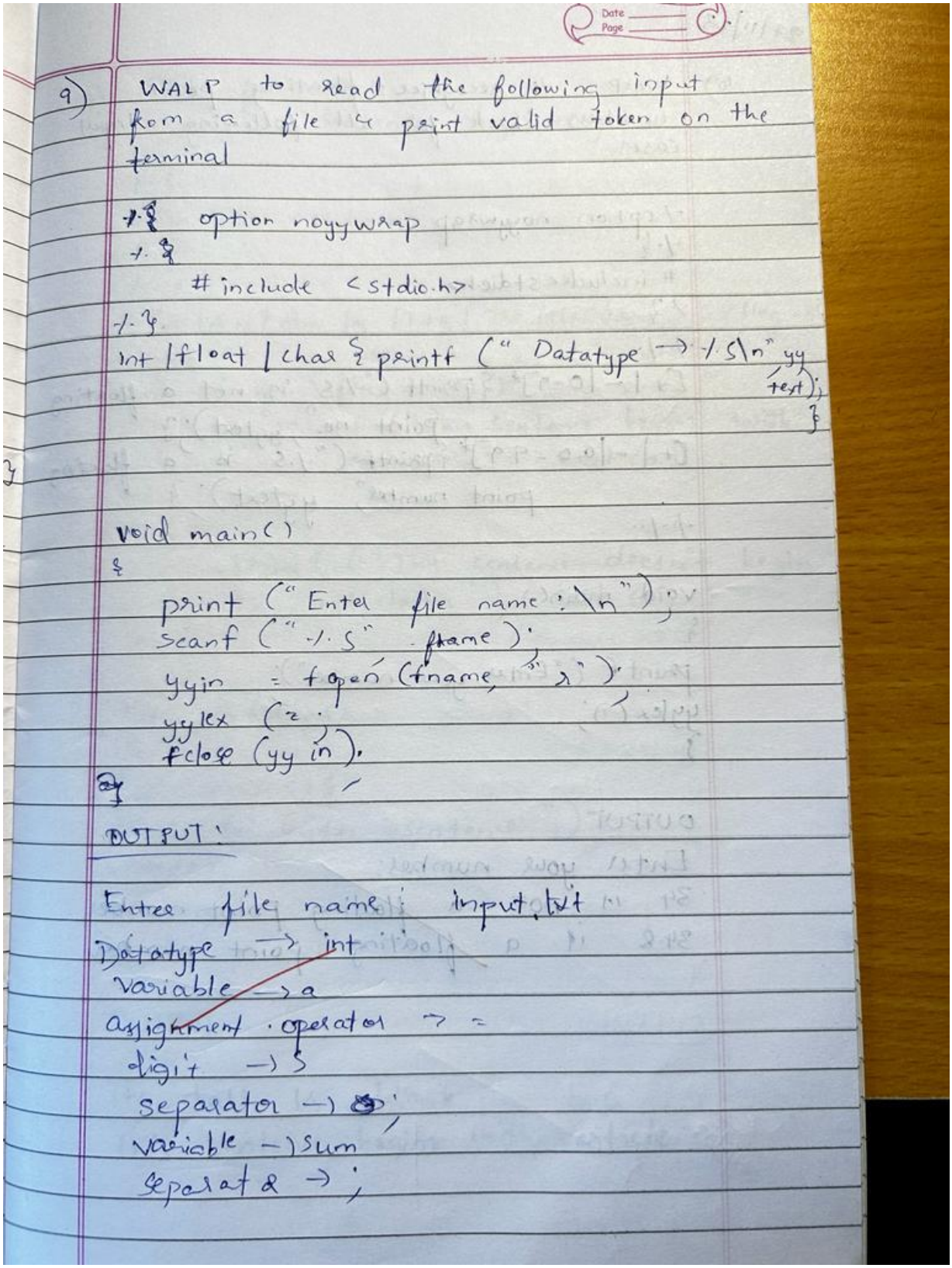
KingKohli
Stream of Characters: KingKohli
digits: 18

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



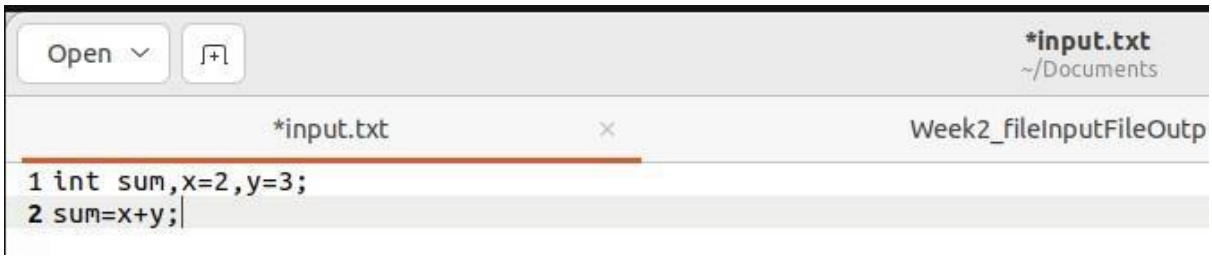
9) WAP to read the following input from a file & print valid token on the terminal

```
%{ option noyywrap
%}
    #include <stdio.h>
%}
int |float |char { printf (" Datatype -> %s\n" yy
                                              text);
                                              }
void main()
{
    print ("Enter file name : \n");
    scanf ("%s" fname);
    yyin = fopen (fname, "r");
    yylex (2.);
    fclose (yy in);
}
```

OUTPUT :

Enter file name : input.txt
Datatype -> int
variable -> a
assignment operator -> =
digit -> 5
separator -> ;
variable -> sum
separator -> ;

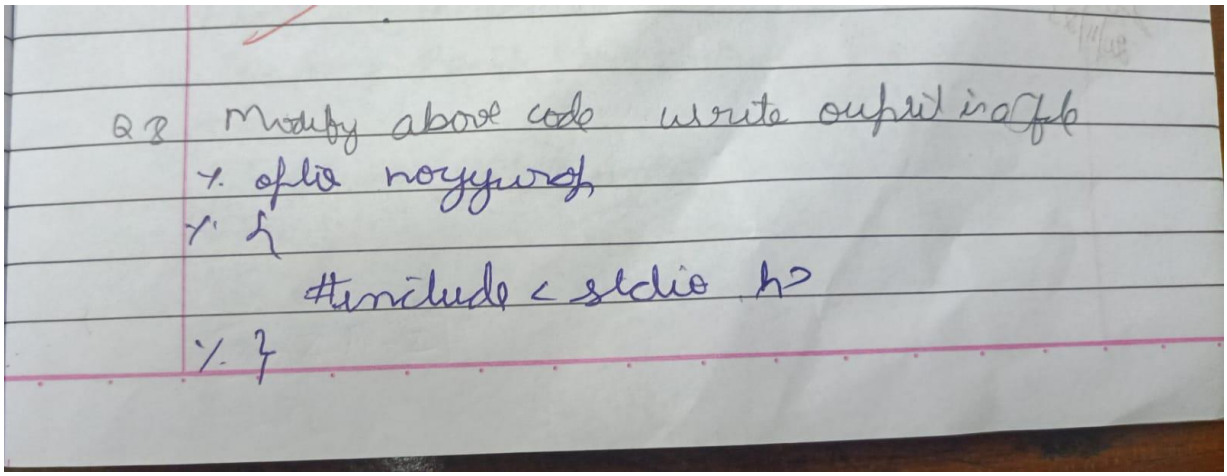
Output

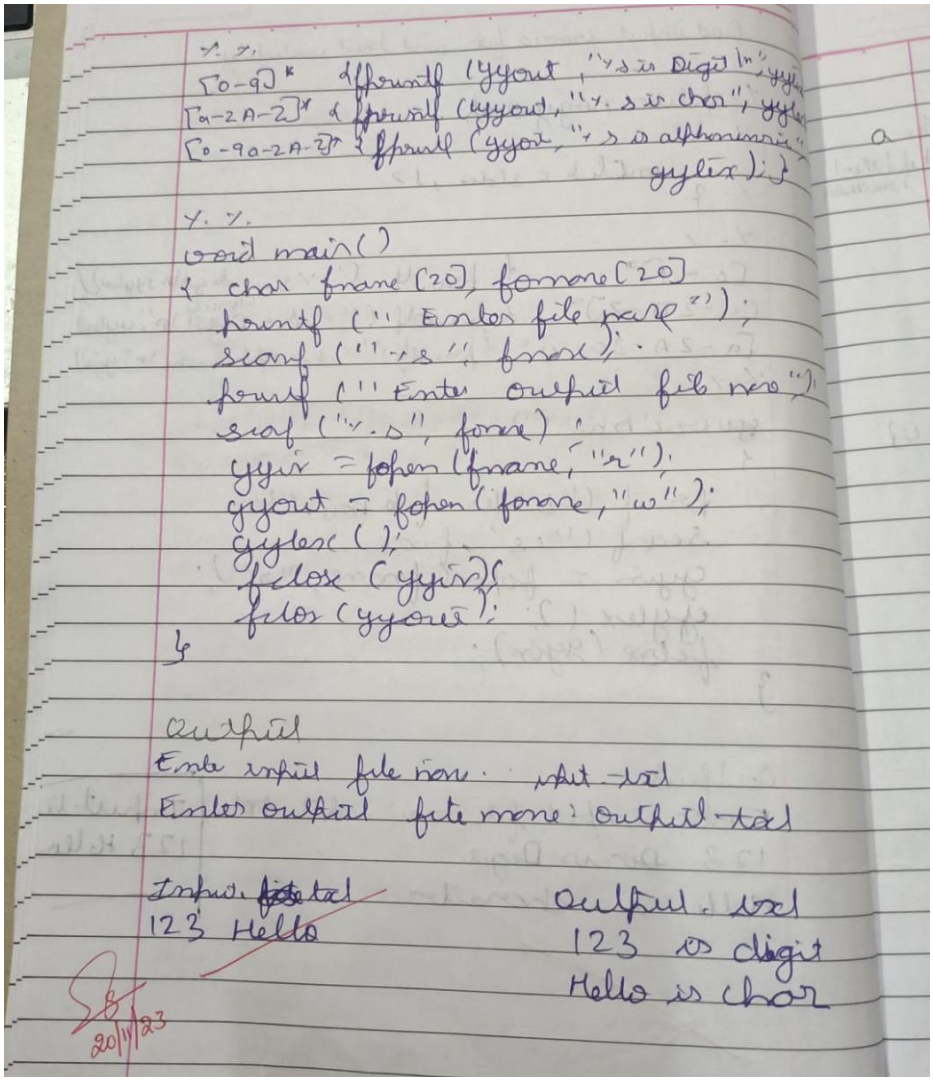



```
int is a keyword.
 sum is an identifier.
, is a separator.
x is an identifier.
= is an assignment operator.
2 is/are digit(s).
, is a separator.
y is an identifier.
= is an assignment operator.
3 is/are digit(s).
; is a delimiter.
sum is an identifier.
= is an assignment operator.
x is an identifier.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

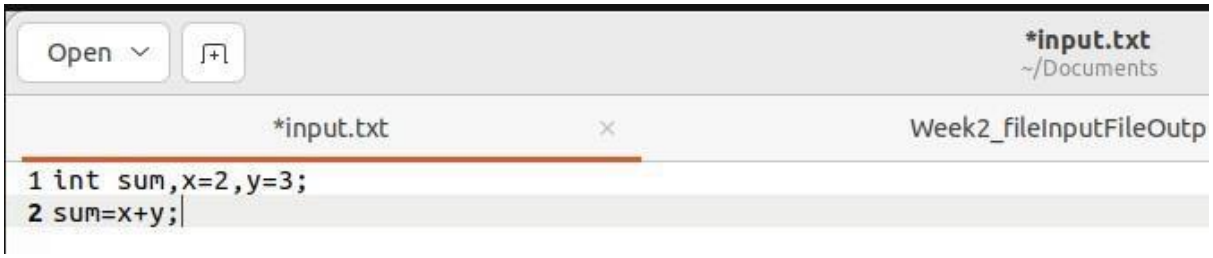
2.4 Write a program in lex to identify tokens in a program by taking input from a file and printing the output in another file.

Code

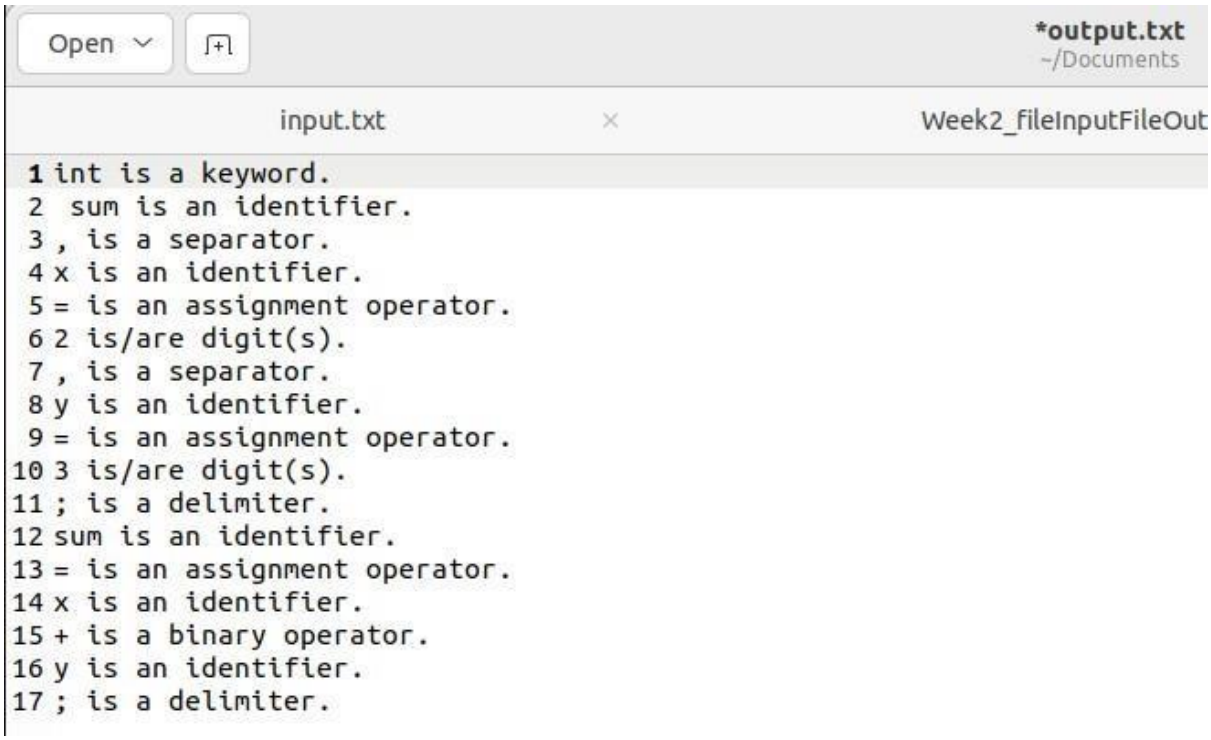




Output



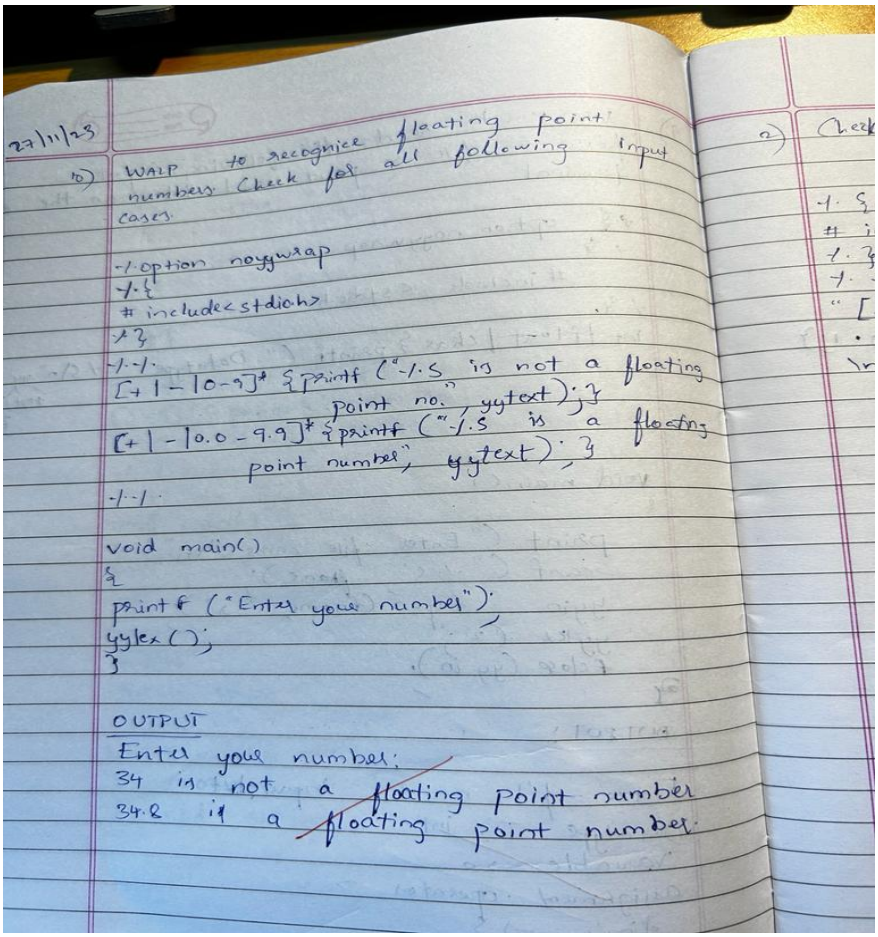

```
Printed in output.txt
```



Lab 3

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

Code



27/11/23

b) WAP to recognize floating point numbers. Check for all following input cases.

```
-/-option noyywrap
-/-{
# include<stdio.h>
-/-}
-/-/-
[+|-]0-9]* {printf ("-/-s is not a floating point no.", yytext);}
[+|-]0.0-9.9]* {printf ("-/-s is a floating point number", yytext);}
-/-/-

void main()
{
printf ("Enter your number");
yylex();
}
```

OUTPUT

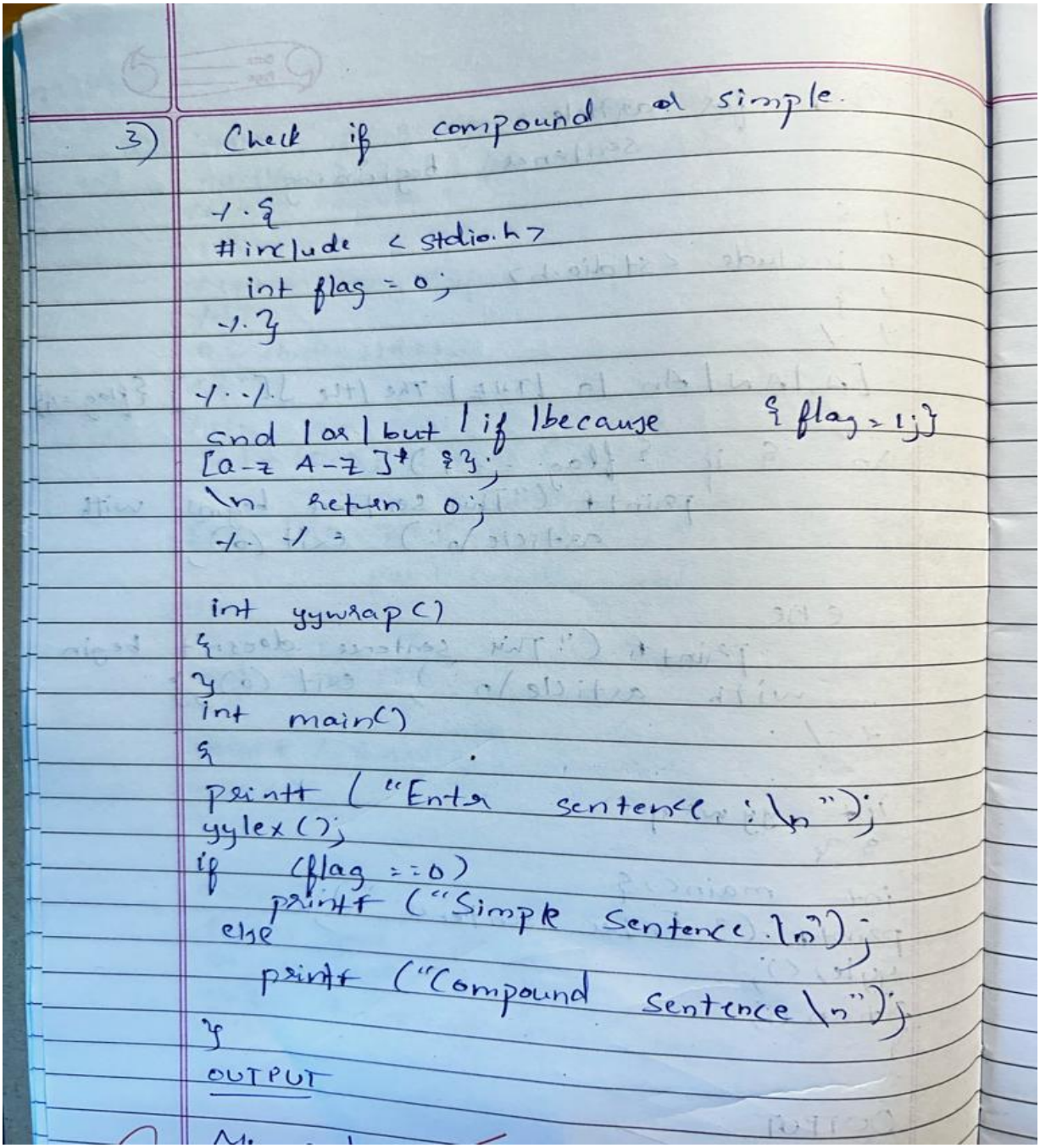
Enter your number:
34 is not a floating point number
34.8 is a floating point number

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



```
3) Check if compound or simple.

%{
#include <stdio.h>
  int flag = 0;
%}

%%
and |or |but |if |because    {flag = 1;}
[a-z A-Z]+ {};
\n  return 0;
%% =

int yywrap()
{
}
int main()
{
printf ("Enter sentence : \n");
yylex();
if (flag == 0)
   printf ("Simple Sentence \n");
else
   printf ("Compound sentence \n");
}
OUTPUT
```

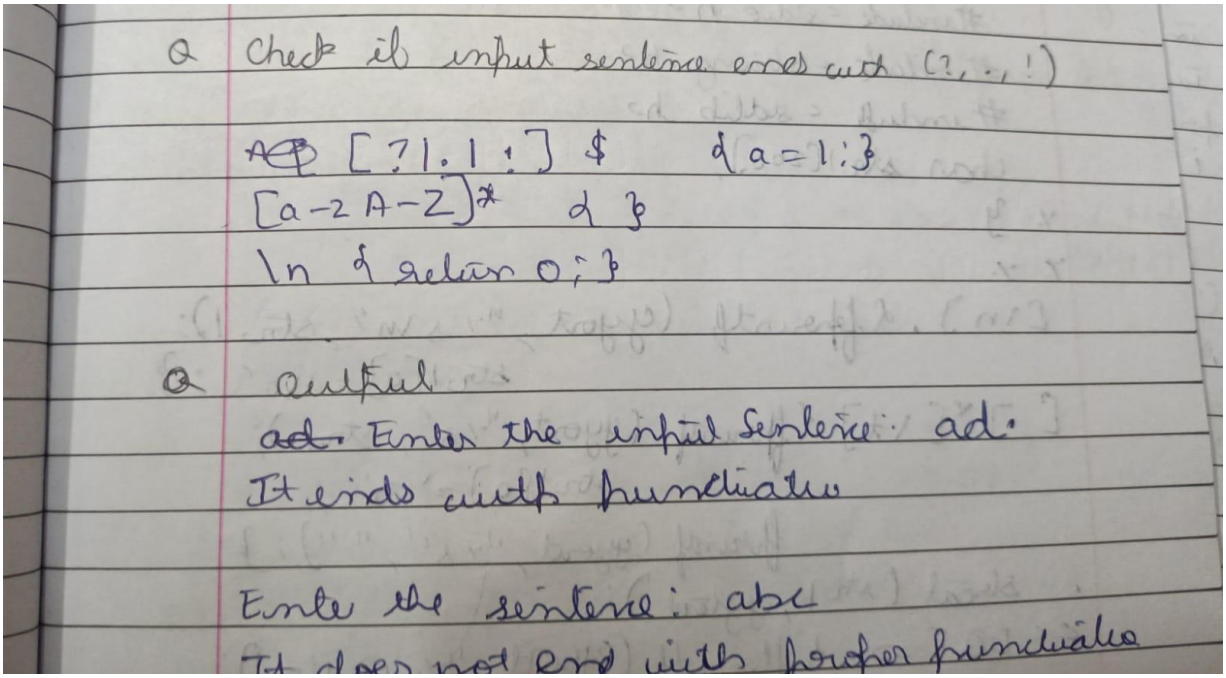
Output

```
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

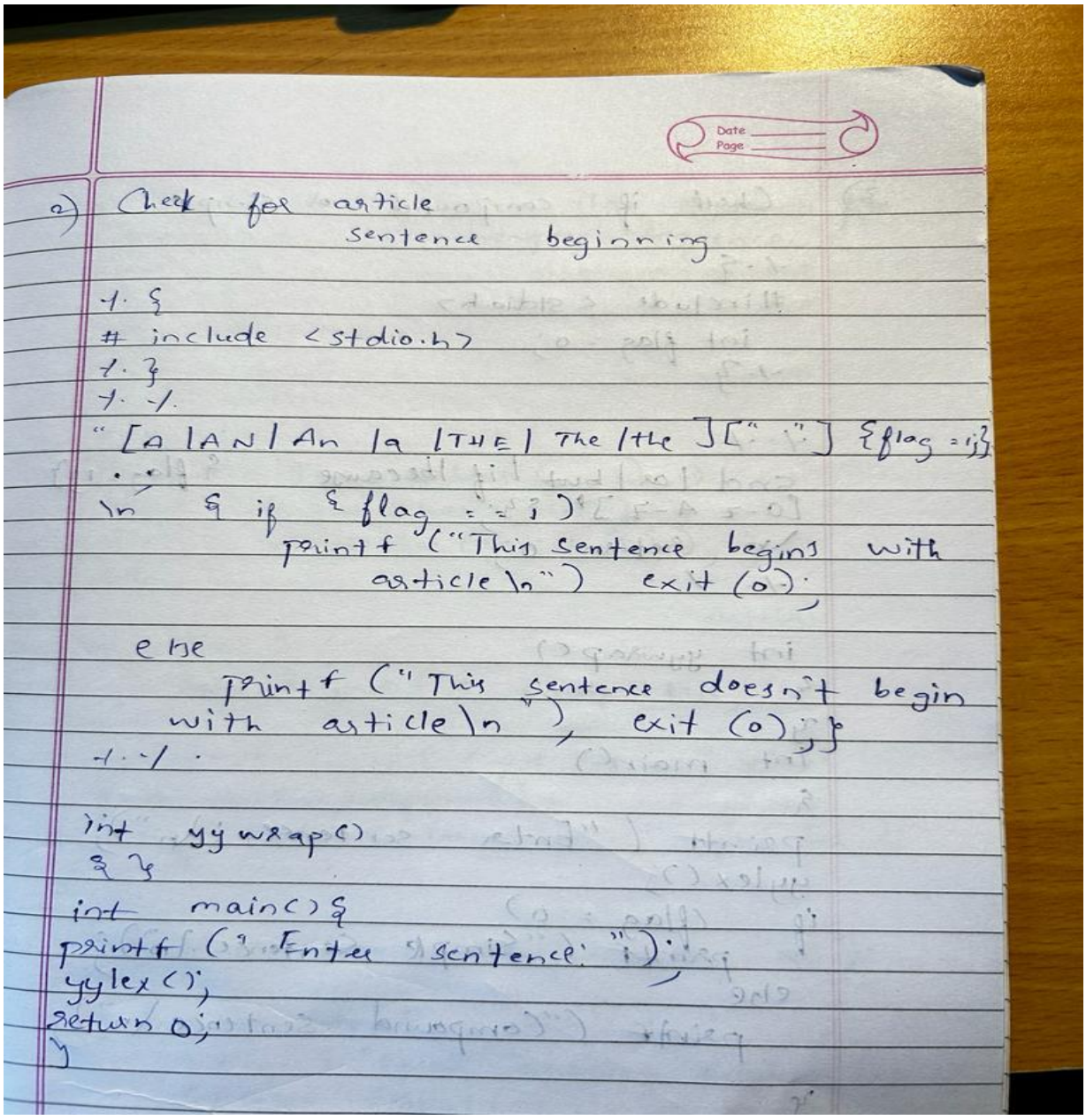
```
Enter a sentence:
Is this yours?
Ends with a punctuation!
```

```
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



```
2) Check for article
        sentence beginning

%{
#include <stdio.h>
%}
%%
"[A |AN| An |a |THE| The |the ][" "] {flag =1;}
.;
\n  { if (flag == 1)
        printf ("This sentence begins with
            article\n") exit (0);

  else
      printf ("This sentence doesn't begin
    with article\n"), exit (0);}
%%

int yywrap()
{}
int main(){
printf ("Enter a sentence: ");
yylex();
return 0;
}
```

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!
```

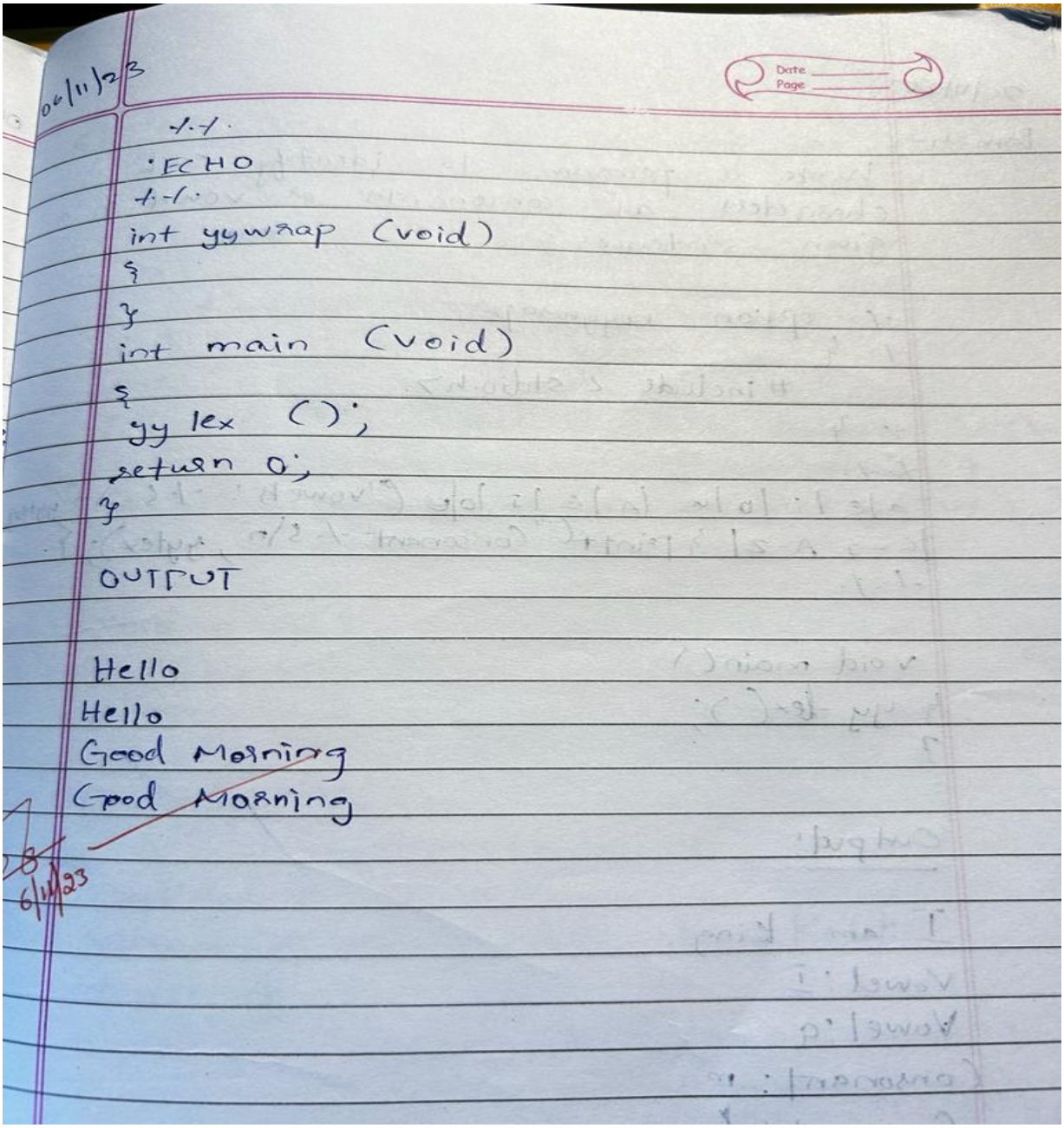
```
Enter a sentence:
The sun rises in the east.
Starts with an article!
```

```
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



```
%%
ECHO
%%
int yywrap (void)
{
}
int main (void)
{
 yy lex ();
 return 0;
}
```

OUTPUT

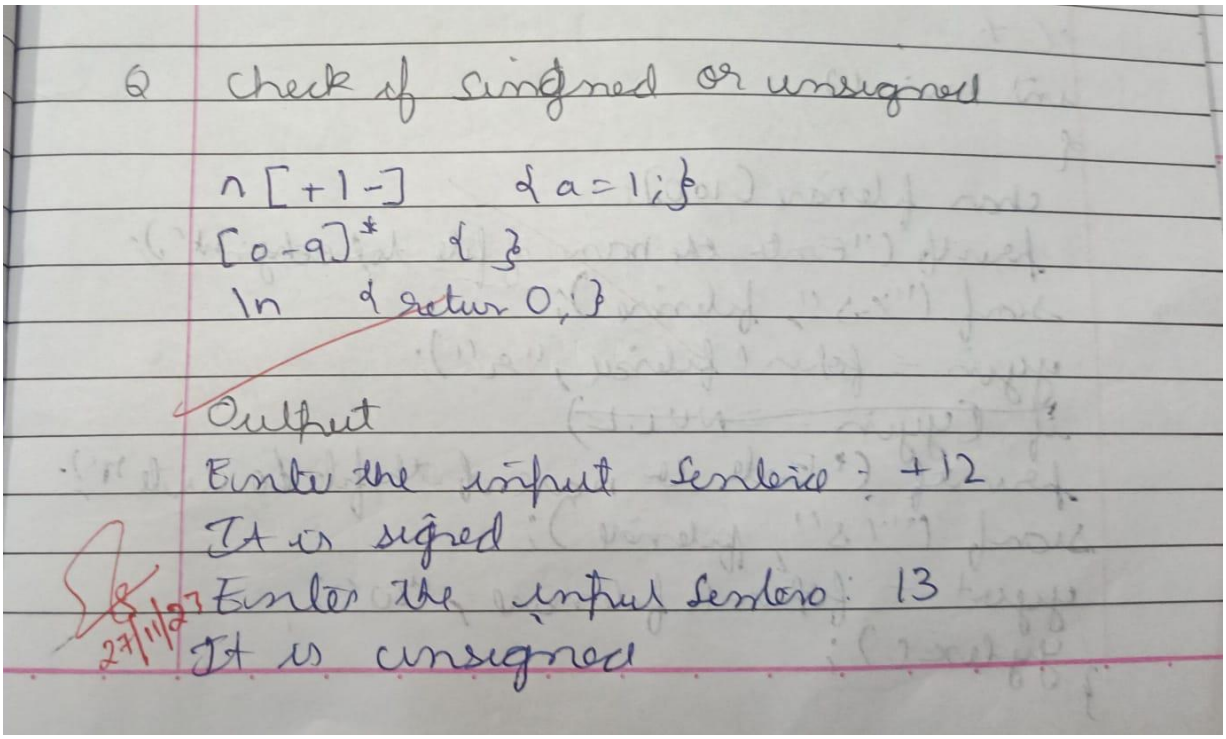
```
Hello
Hello
Good Morning
Good Morning
```

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



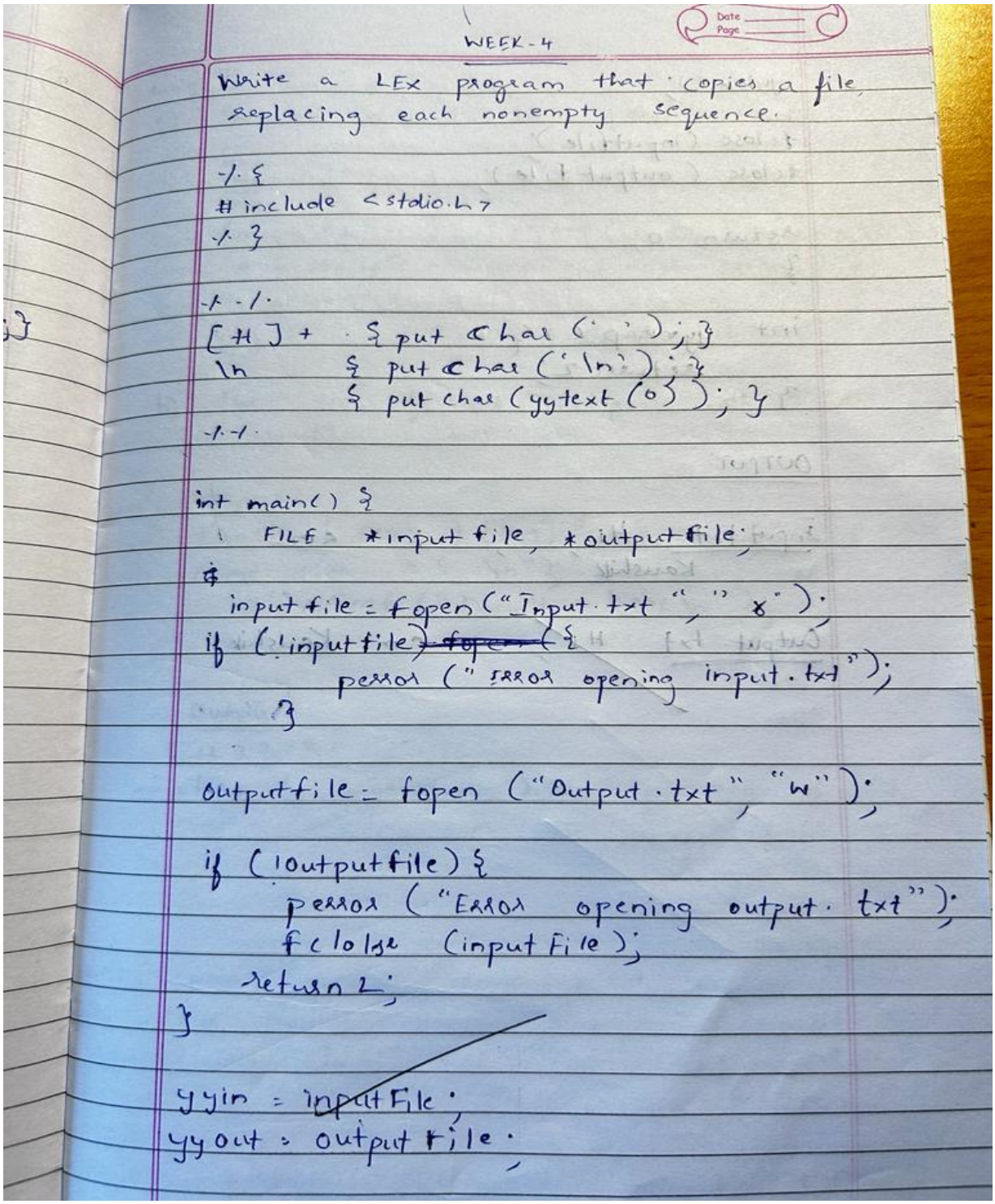
Output

```
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.

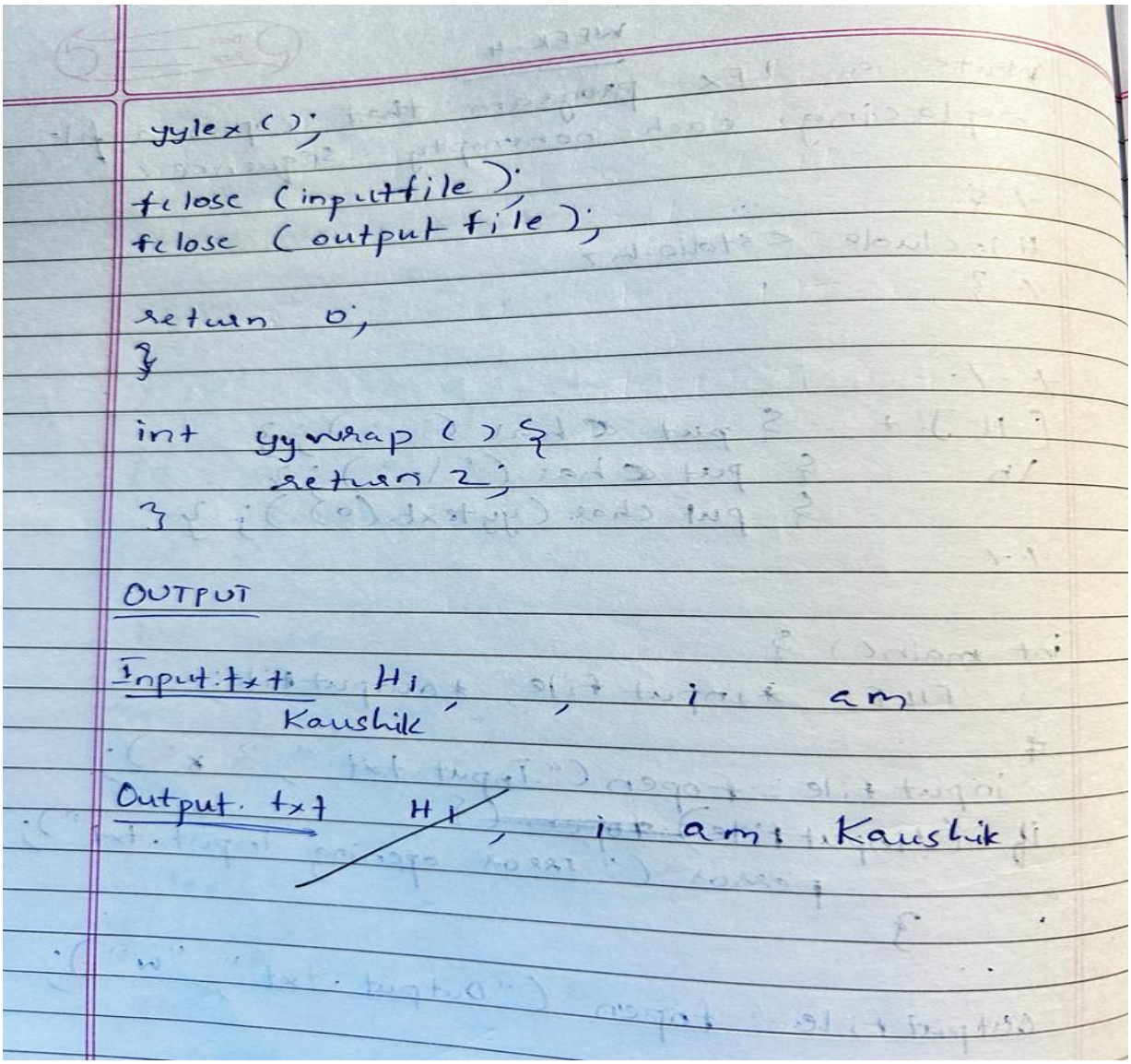
Code



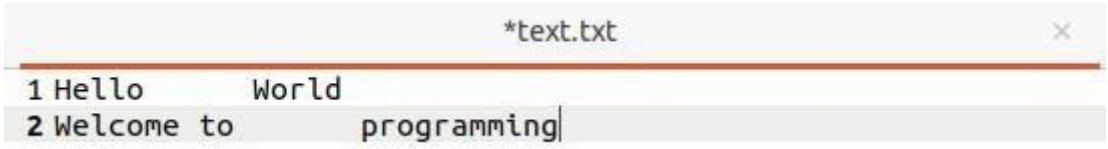
WEEK-4

Write a LEX program that copies a file, replacing each nonempty sequence.

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

%%
[ \t]+      { putchar(' '); }
\n          { putchar('\n'); }
.           { putchar(yytext[0]); }
%%

int main() {
    FILE *inputfile, *outputfile;

    inputfile = fopen("Input.txt", "r");
    if (!inputfile) {
        perror("Error opening input.txt");
    }

    outputfile = fopen("Output.txt", "w");

    if (!outputfile) {
        perror("Error opening output.txt");
        fclose(inputFile);
        return 2;
    }

    yyin = inputFile;
    yyout = outputfile;
```


yylex();
fclose(inputfile);
fclose(outputfile);
return 0;
}
int yywrap(){
return 2;
}
OUTPUT
Input.txt Hi, i am Kaushik
Output.txt Hi, i am Kaushik

Output


*text.txt
1 Hello World
2 Welcome to programming

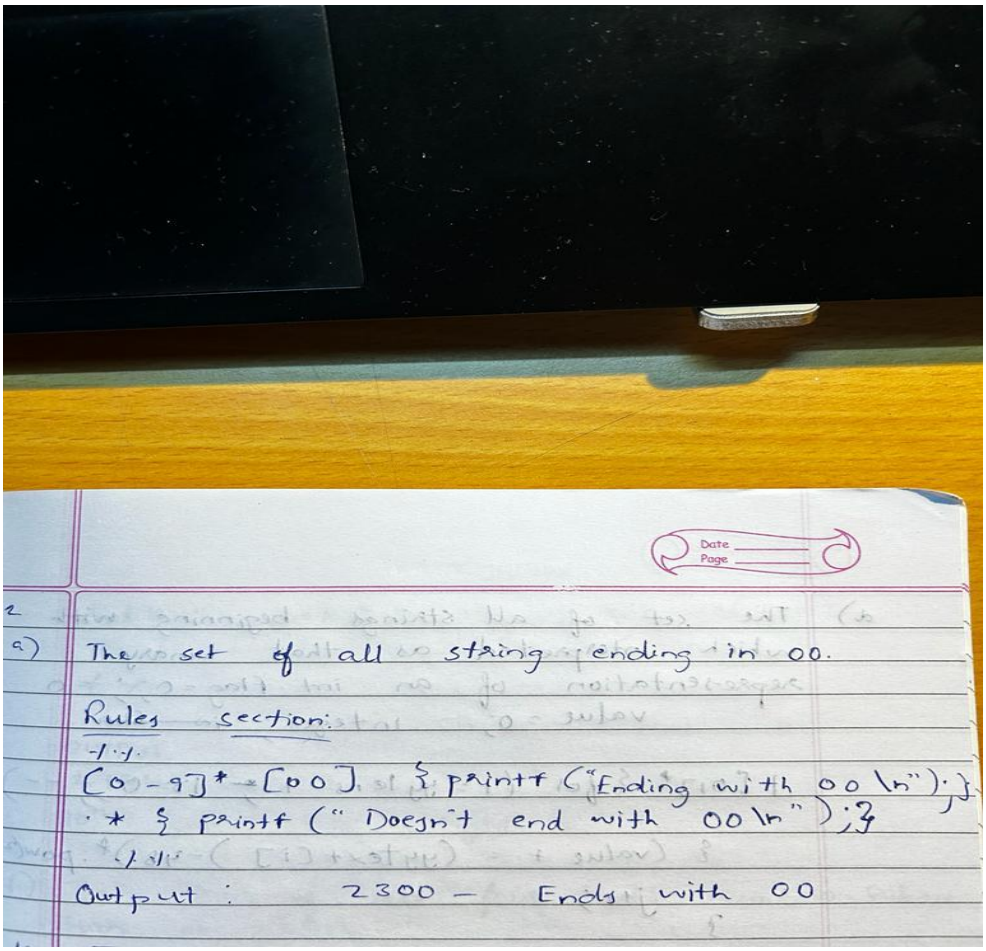

Printed!


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



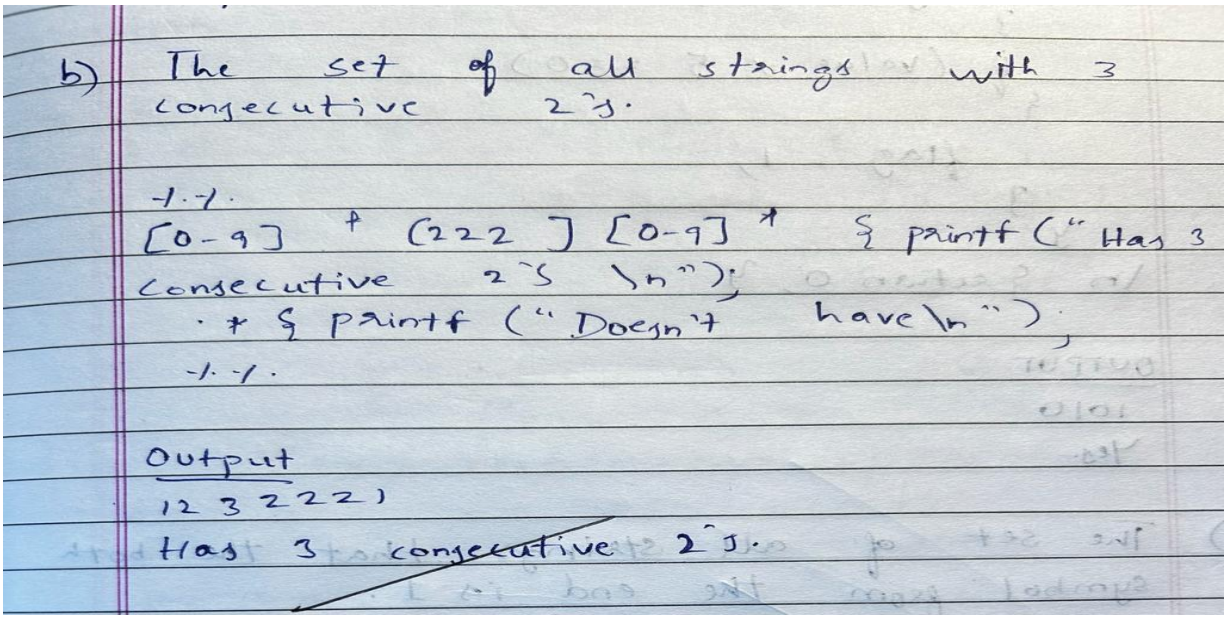
Output

```
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



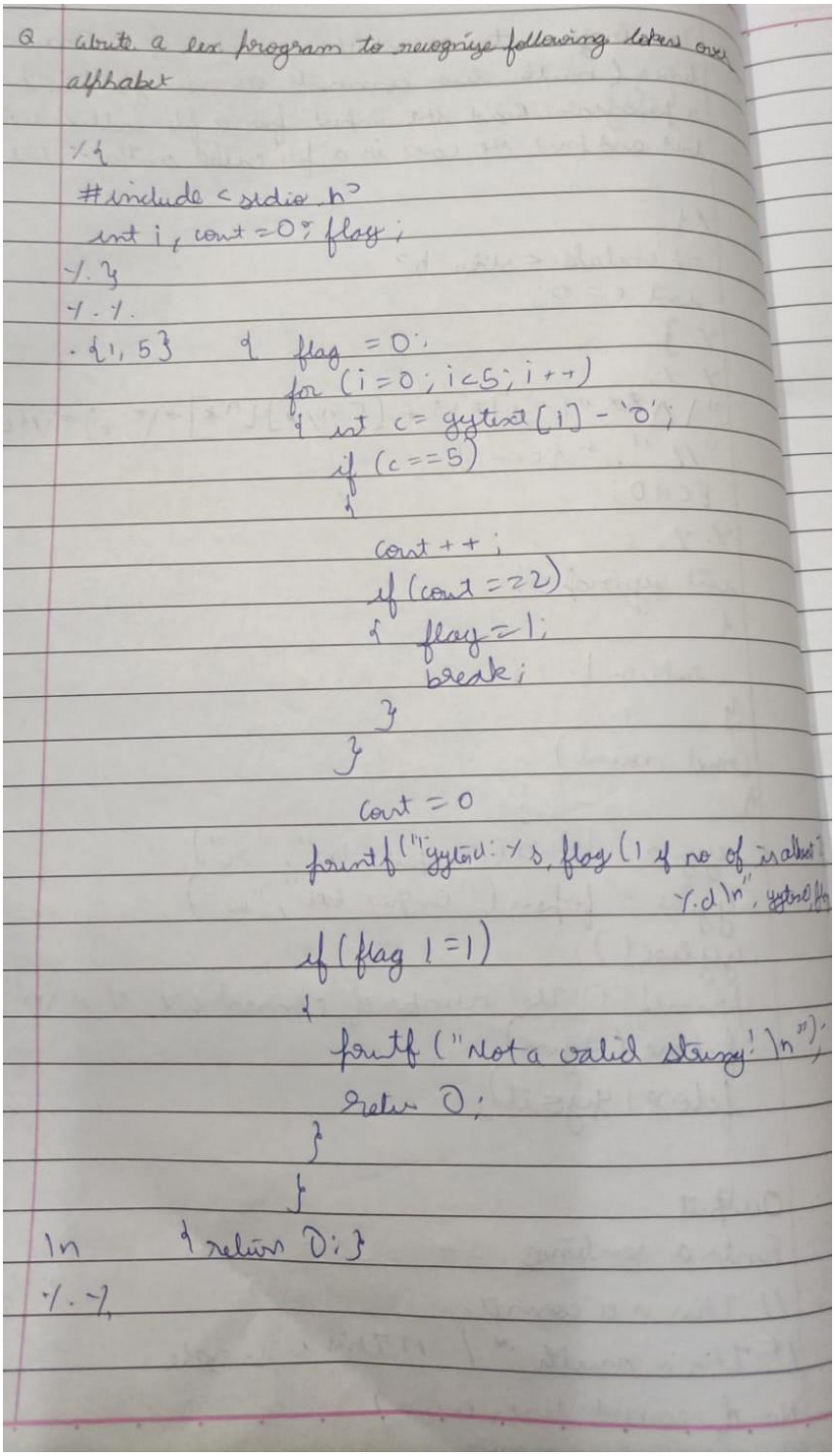
Output

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

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

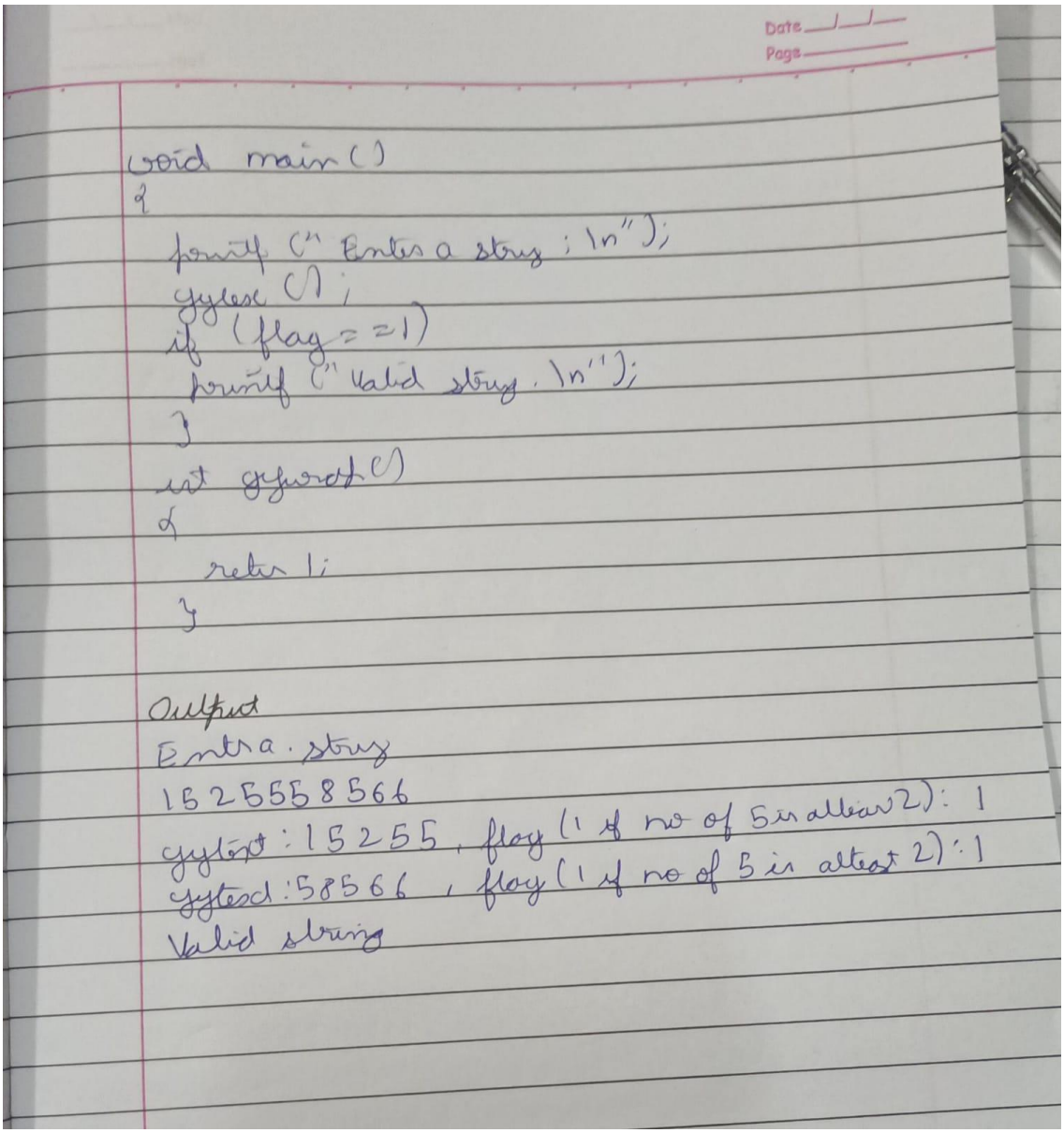
4.2.3 The set of all string such that every block of five consecutive symbols contains at least two 5's.

Code



Q write a lex program to recognize following letters over alphabet

```
%{
#include <stdio.h>
int i, cont=0; flag;
%}
%%
.{1,5}  { flag=0;
          for(i=0;i<5;i++)
          { int c= yytext[i] - '0';
            if(c==5)
            {
              cont++;
              if(cont==2)
              { flag=1;
                break;
              }
            }
            cont=0
          fprintf("yytext: %s, flag(1 if no of is also
                                    %d\n", yytext[i]
          if(flag!=1)
          {
            fprintf("Not a valid string\n");
            return 0;
          }
          }
\n        { return 0;}
%%
```

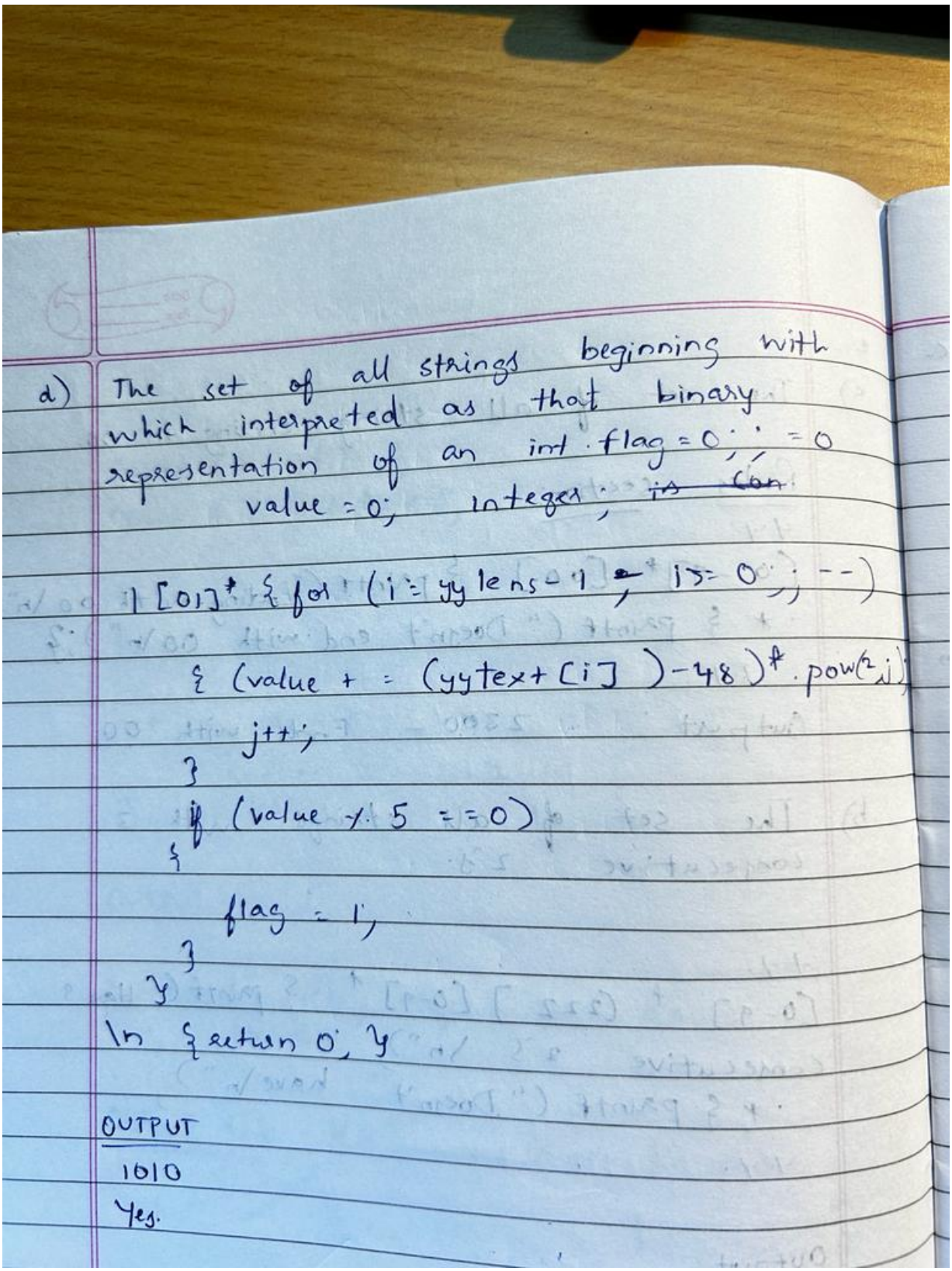


Output

```
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.
```

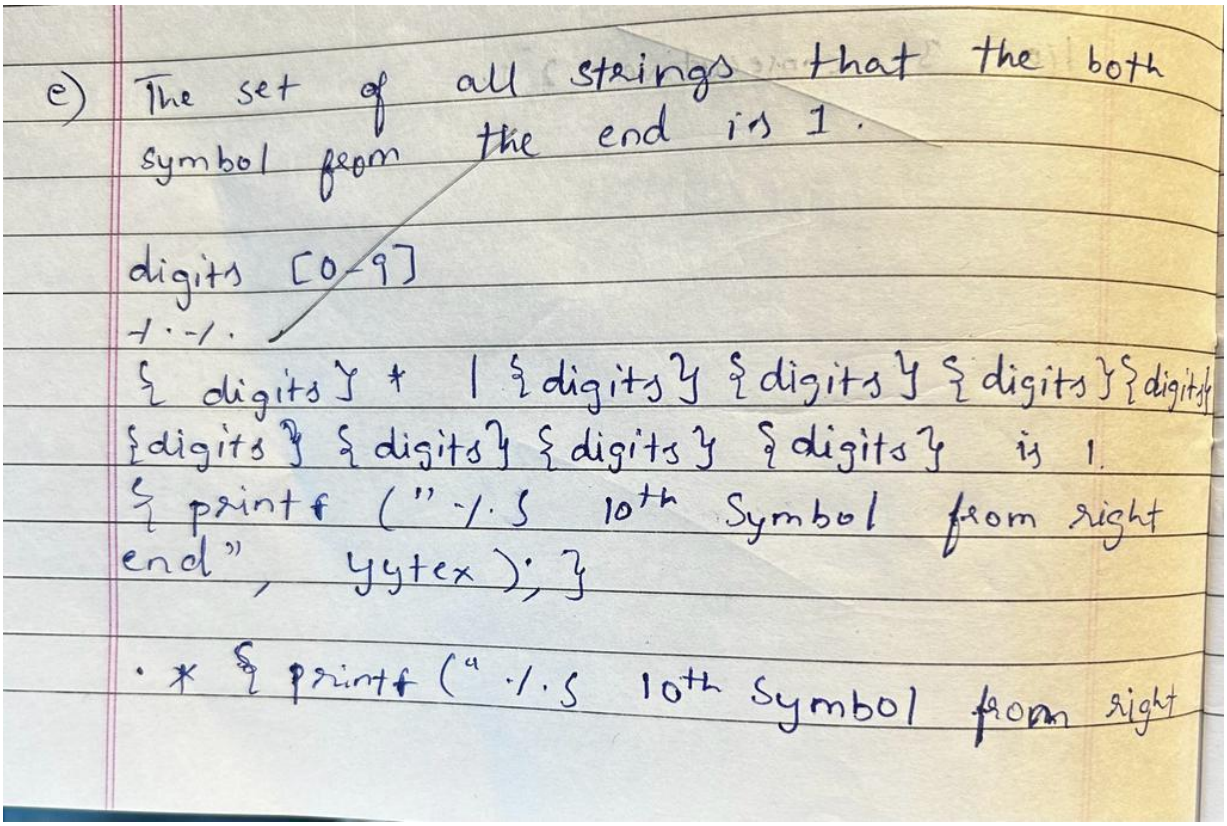
```
Enter a string:
101
Decimal representation:5
Congruent to modulo 5.
```

```
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

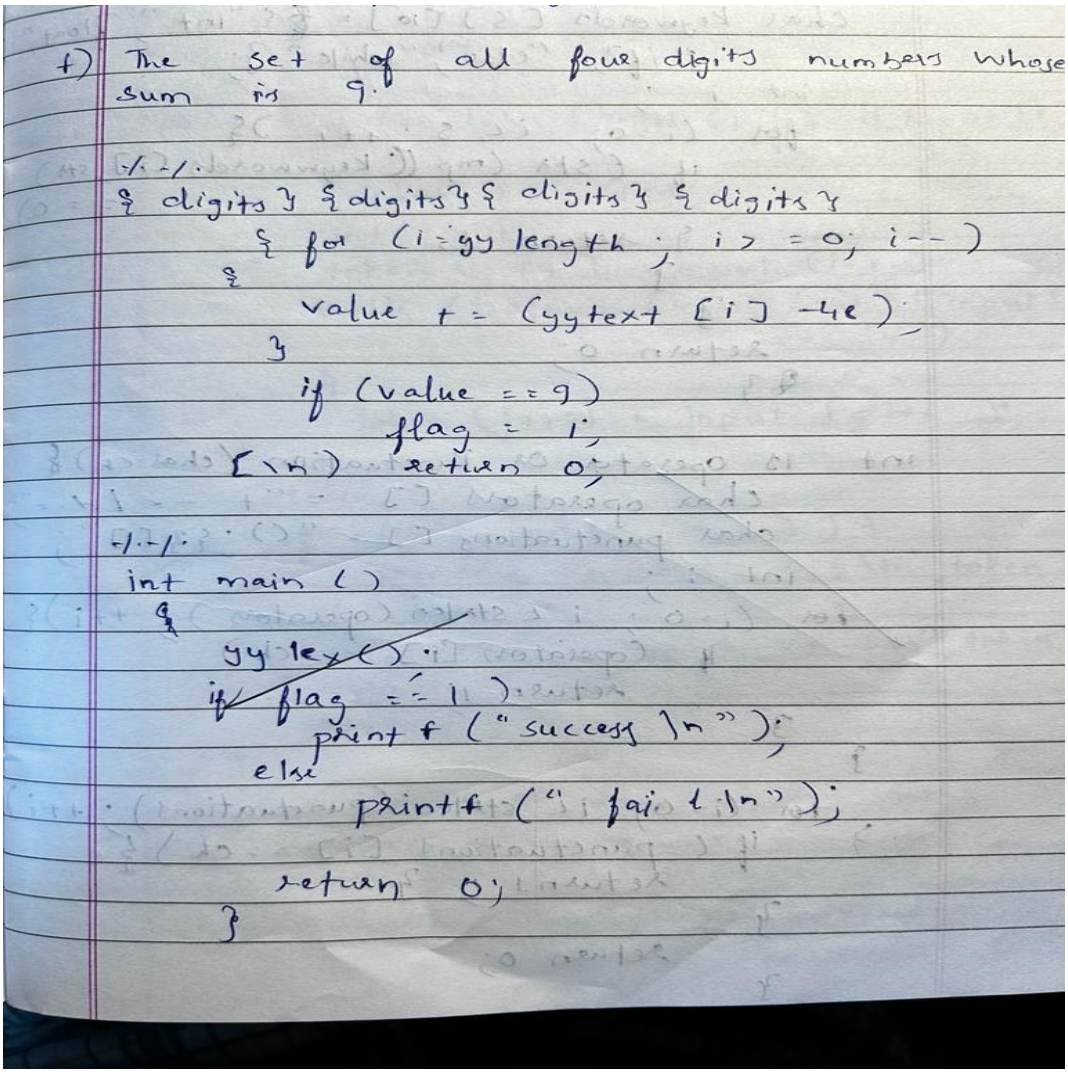


Output

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

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

Code



```
f) The set of all four digits numbers whose
sum is 9.

%-%
% digits % % digits % % digits % % digits %
    % for (i=yy length ; i >= 0; i--)
    %
        value += (yytext [i] -4e);
    }
    if (value == 9)
        flag = 1;
[vm)    retun 0;

%-%
int main ()
%
    yy lex ();
    if flag == 1 );
        print f ("success \n");
    else
        printf ("fai t \n");
    retun 0;
}
```

Output

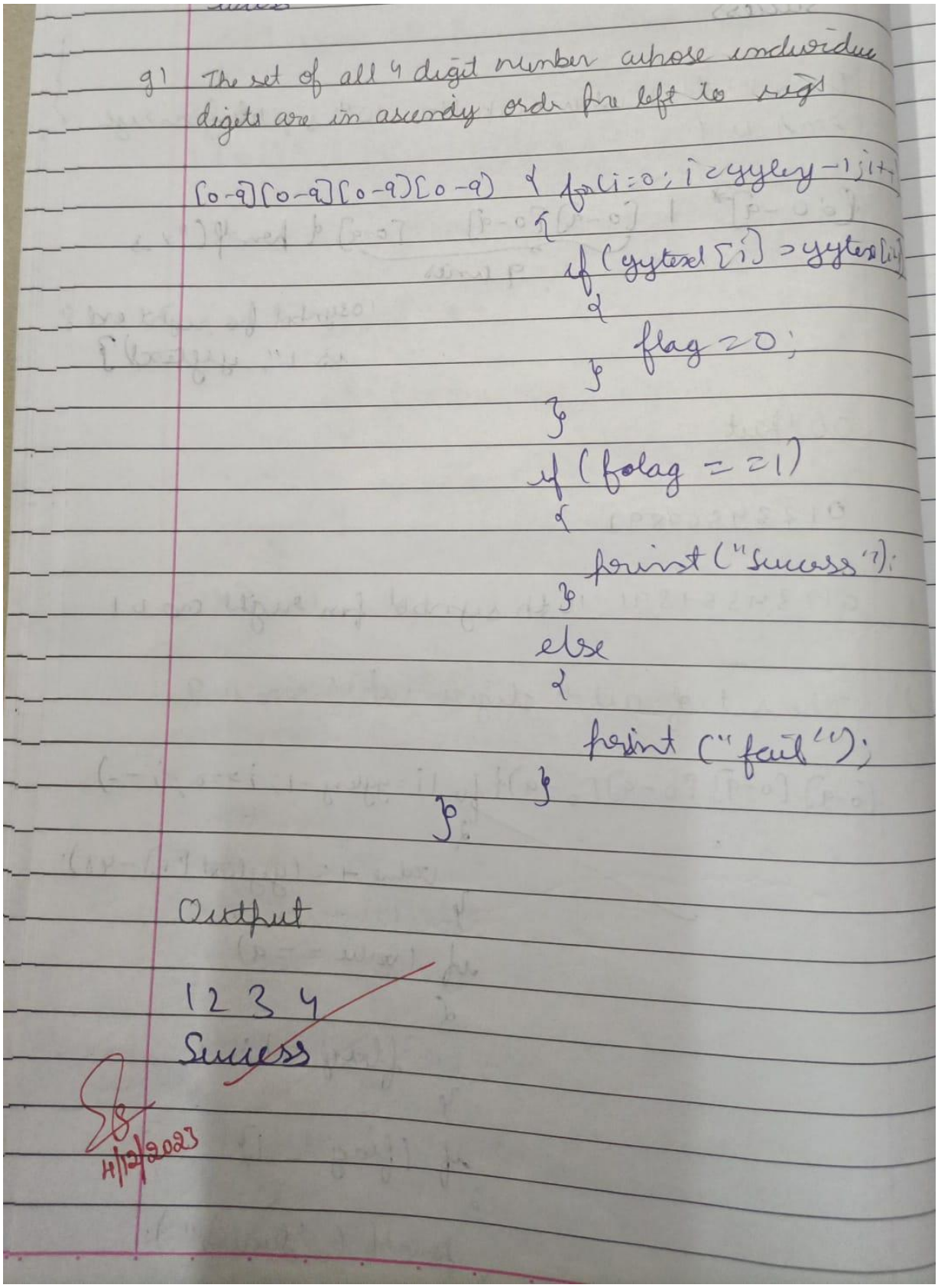
```
Enter a string:
6300
The sum of digits is 9.
```

```
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

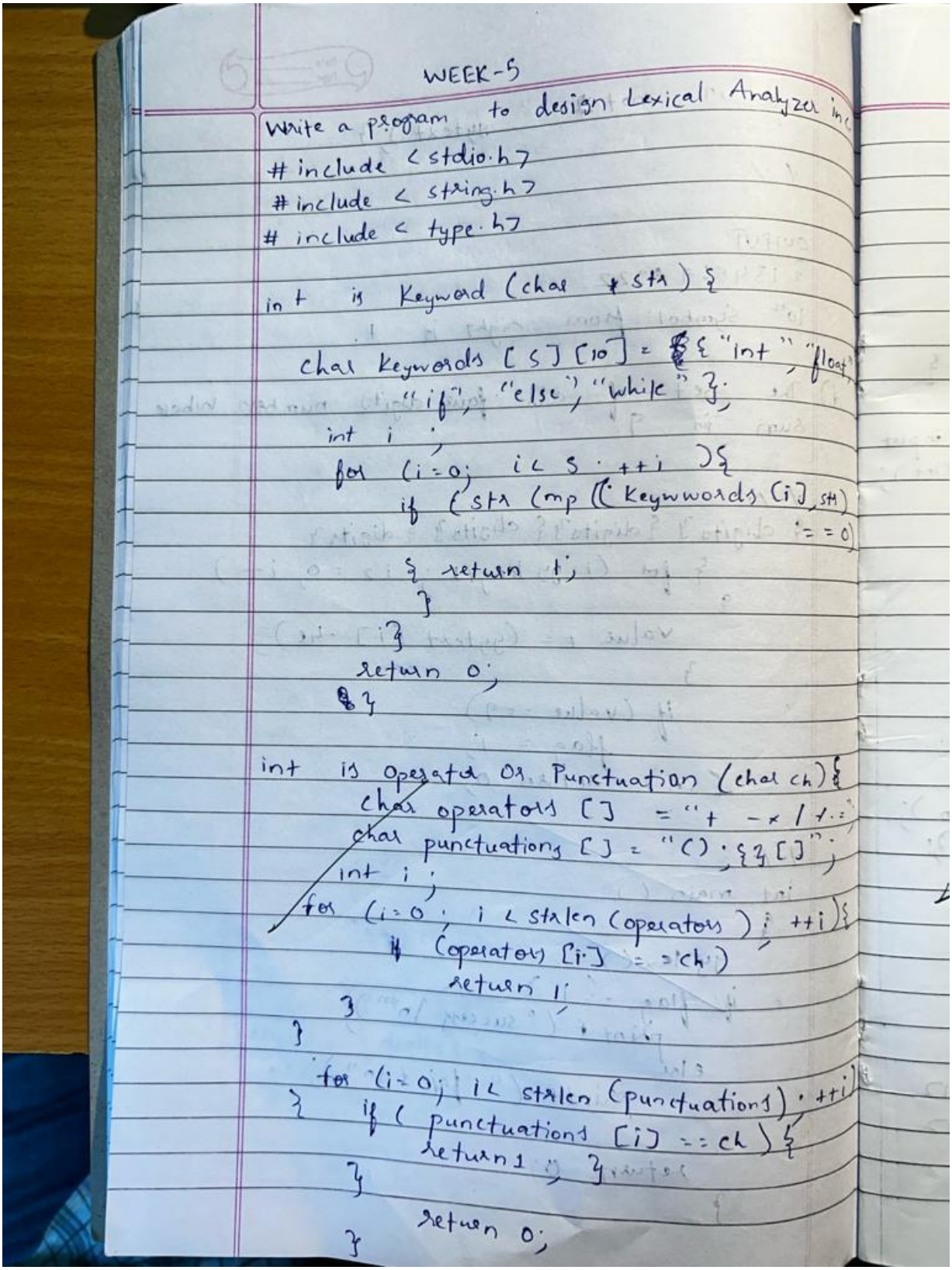
```
Enter a string:
1235
The digits are in ascending order.
```

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

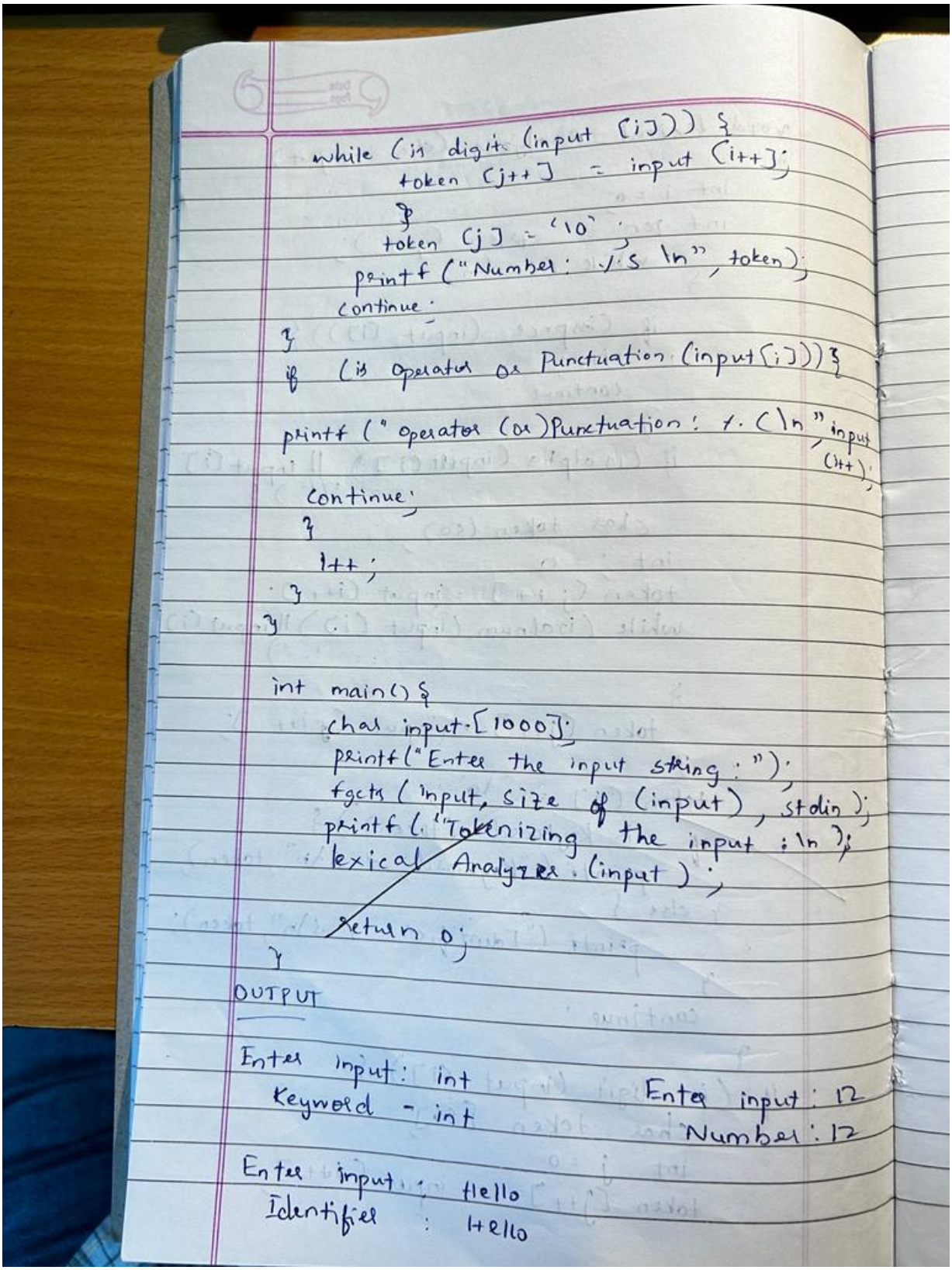
Lab 5

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

Code



```
WEEK-5
Write a program to design Lexical Analyzer in C
#include <stdio.h>
#include <string.h>
#include <type.h>

int isKeyword(char *str) {
    char keywords[5][10] = {"int", "float",
        "if", "else", "while"};
    int i;
    for (i=0; i<5; ++i) {
        if (strcmp(keywords[i], str)
            == 0)
        { return 1;
        }
    }
    return 0;
}

int isOperator Or Punctuation (char ch){
    char operators[] = "+ - * / =";
    char punctuations[] = "(), {}[]";
    int i;
    for (i=0; i<strlen(operators); ++i){
        if (operators[i] == ch)
            return 1;
    }
}
    for (i=0; i<strlen(punctuations); ++i)
    if (punctuations[i] == ch) {
        return 1; }
    }
    return 0;
}
```



```
while (is digit (input [i])) {
    token [j++] = input [i++];
}
token [j] = '\0';
printf ("Number: %s \n", token);
continue;
}
if (is operator or Punctuation (input[i])){

printf ("operator (or) Punctuation: %c \n", input[i++]);
continue;
}
i++;
}
}

int main(){
char input[1000];
printf("Enter the input string:");
fgets (input, sizeof (input), stdin);
printf ("Tokenizing the input: \n");
lexical Analyzer (input);
return 0;
}
```

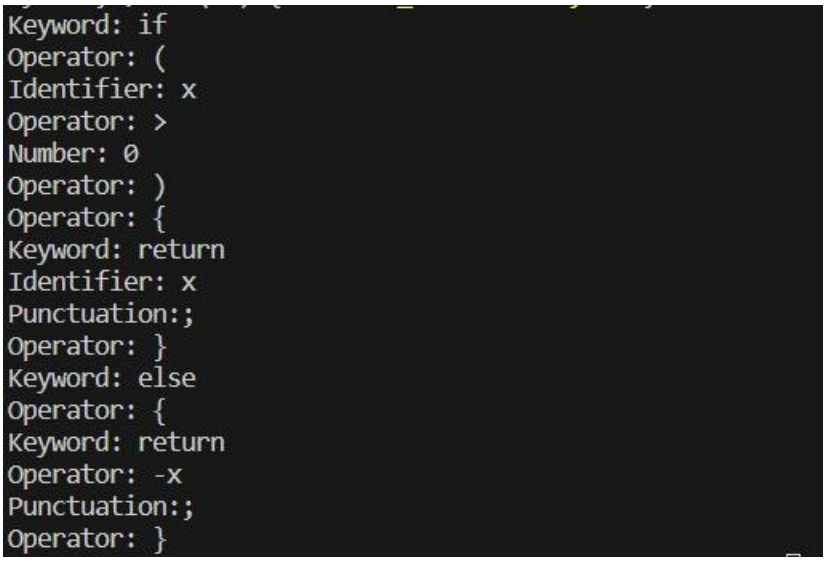
OUTPUT

Enter input: int
Keyword - int

Enter input: 12
Number: 12

Enter input: Hello
Identifier: Hello

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: }

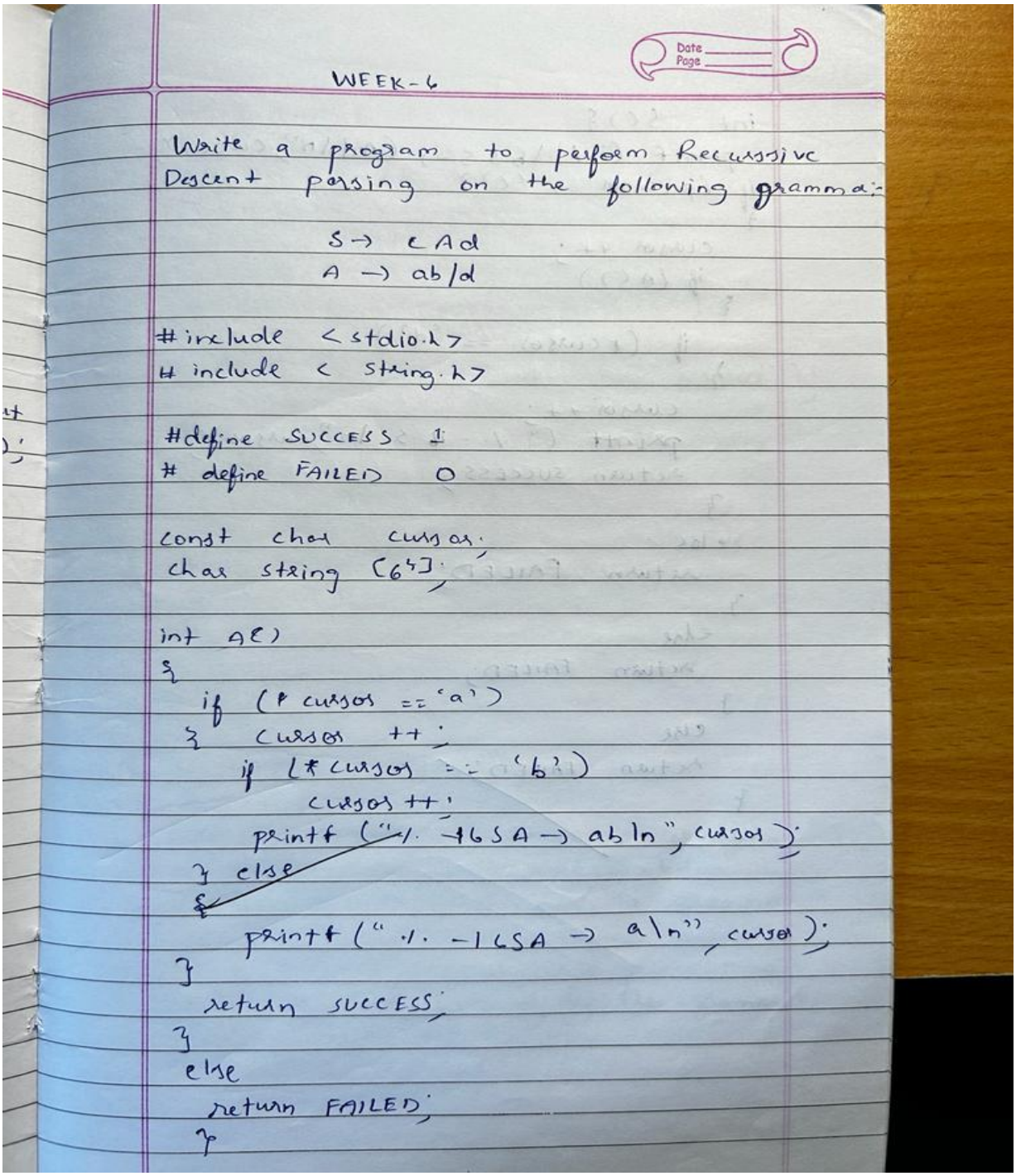
Lab 6

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

S->cAd

A->ab | a

Code

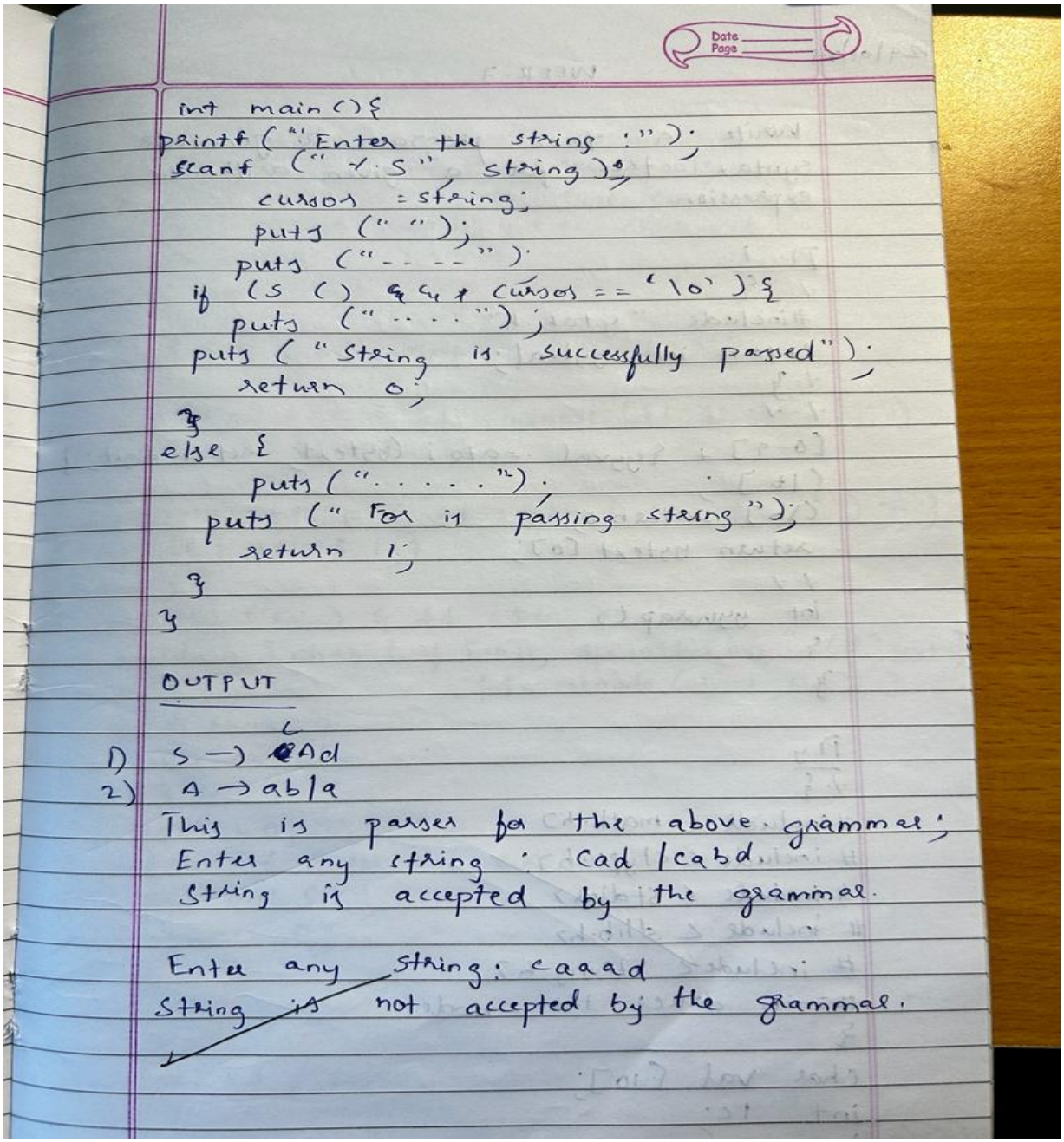


WEEK-6

Write a program to perform Recursive Descent parsing on the following grammar:

S → cAd
A → ab/d

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

#define SUCCESS 1
#define FAILED 0

const char cursor;
char string[64];

int A()
{
  if (*cursor == 'a')
  {
    cursor++;
    if (*cursor == 'b')
      cursor++;
    printf("\n. -1 6SA -> ab\n", cursor);
  } else
  {
    printf("\n. -1 6SA -> a\n", cursor);
  }
  return SUCCESS;
}
else
  return FAILED;
}
```



Output

```
Enter a string:
cad$
Valid string!
```

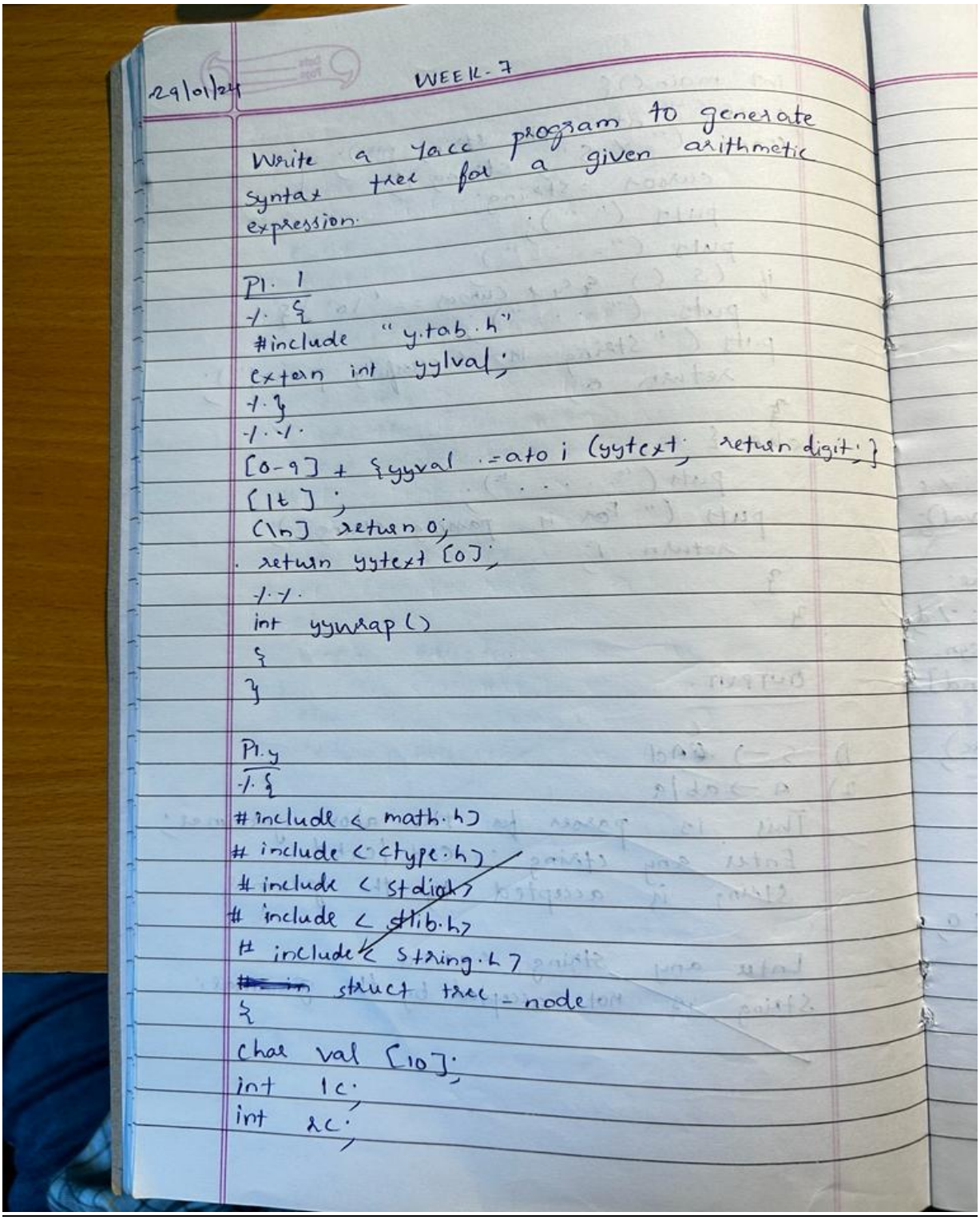
```
Enter a string:
caad$
Invalid String!
```

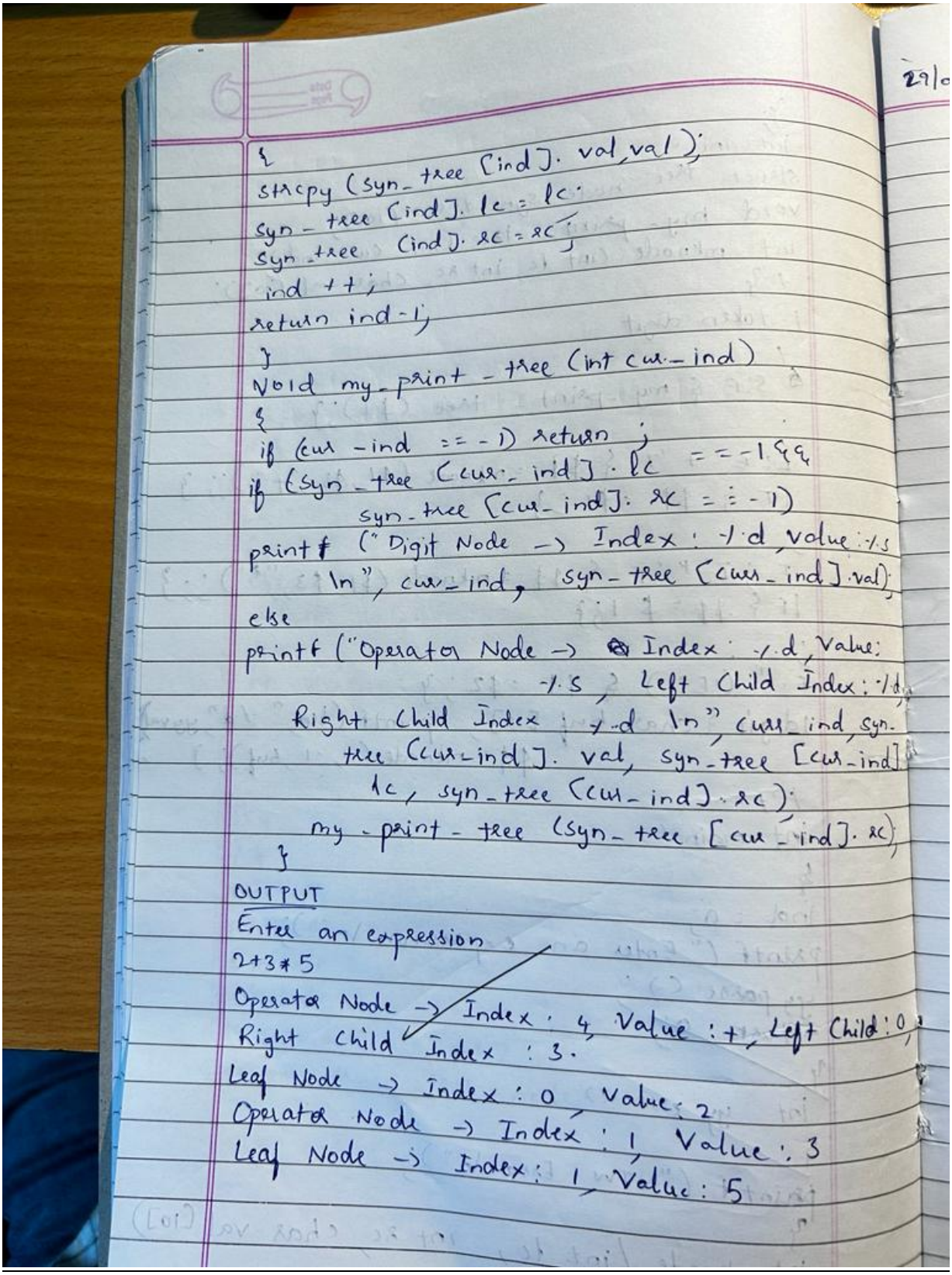
```
Enter a string:
cabd$
Valid string!
```

Lab 7

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

Code





```
{
strcpy (syn_tree [ind]. val, val);
syn_tree [ind]. lc = lc;
syn_tree [ind]. rc = rc;
ind ++;
return ind-1;
}
Void my_print_tree (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, syn_tree [cur_ind]. val);
else
printf ("Operator Node -> Index : %d, Value:
    %s, Left Child Index : %d,
  Right Child Index : %d \n", cur_ind, syn.
    tree [cur_ind]. val, syn_tree [cur_ind].
    lc, syn_tree [cur_ind]. rc);
  my_print_tree (syn_tree [cur_ind]. rc);
}
```

OUTPUT

Enter an expression

2+3*5

Operator Node -> Index : 4, Value : +, Left Child : 0,
Right Child Index : 3.
Leaf Node -> Index : 0, Value : 2
Operator Node -> Index : 1, Value : 3
Leaf Node -> Index : 1, Value : 5

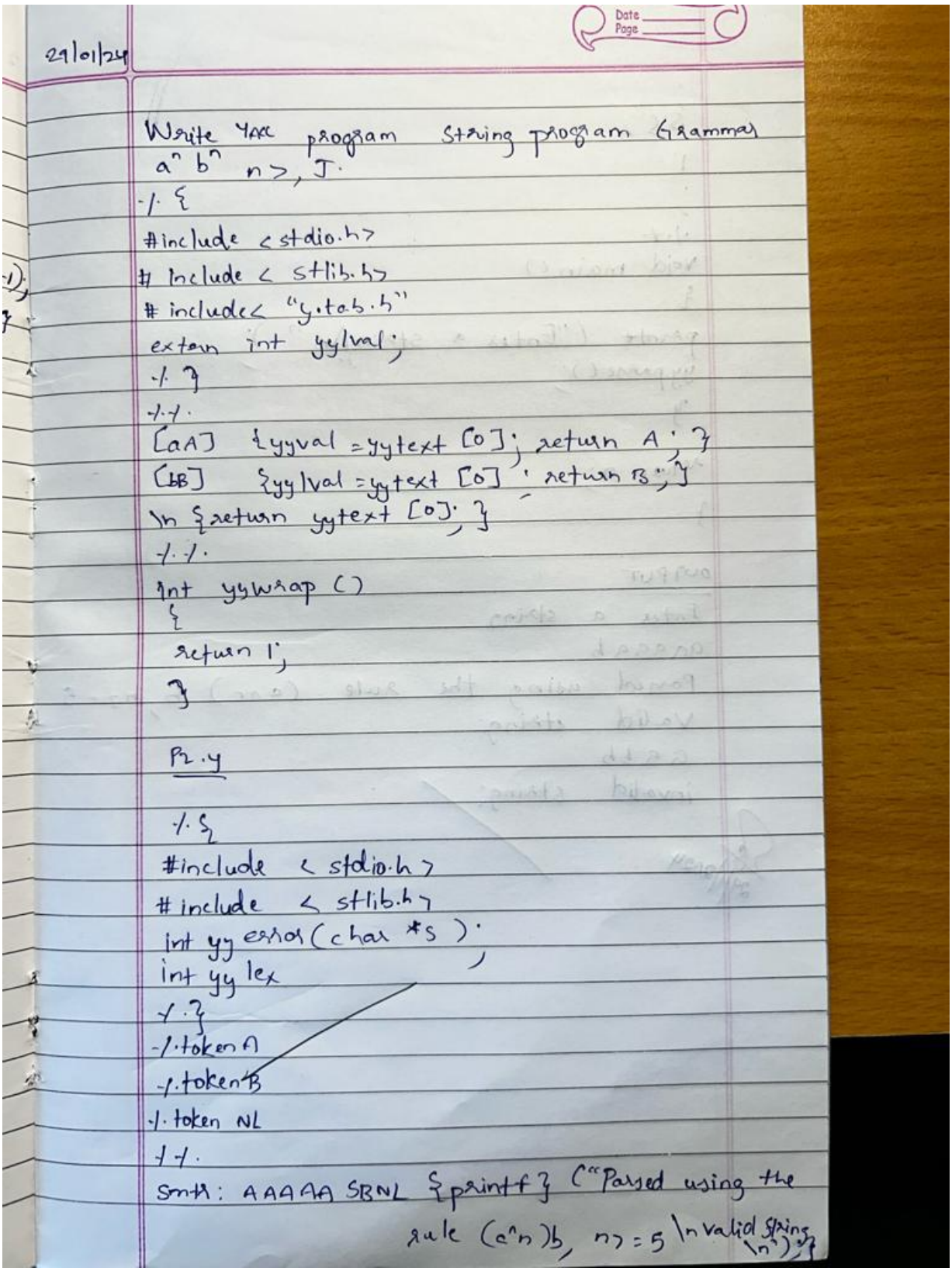
Output

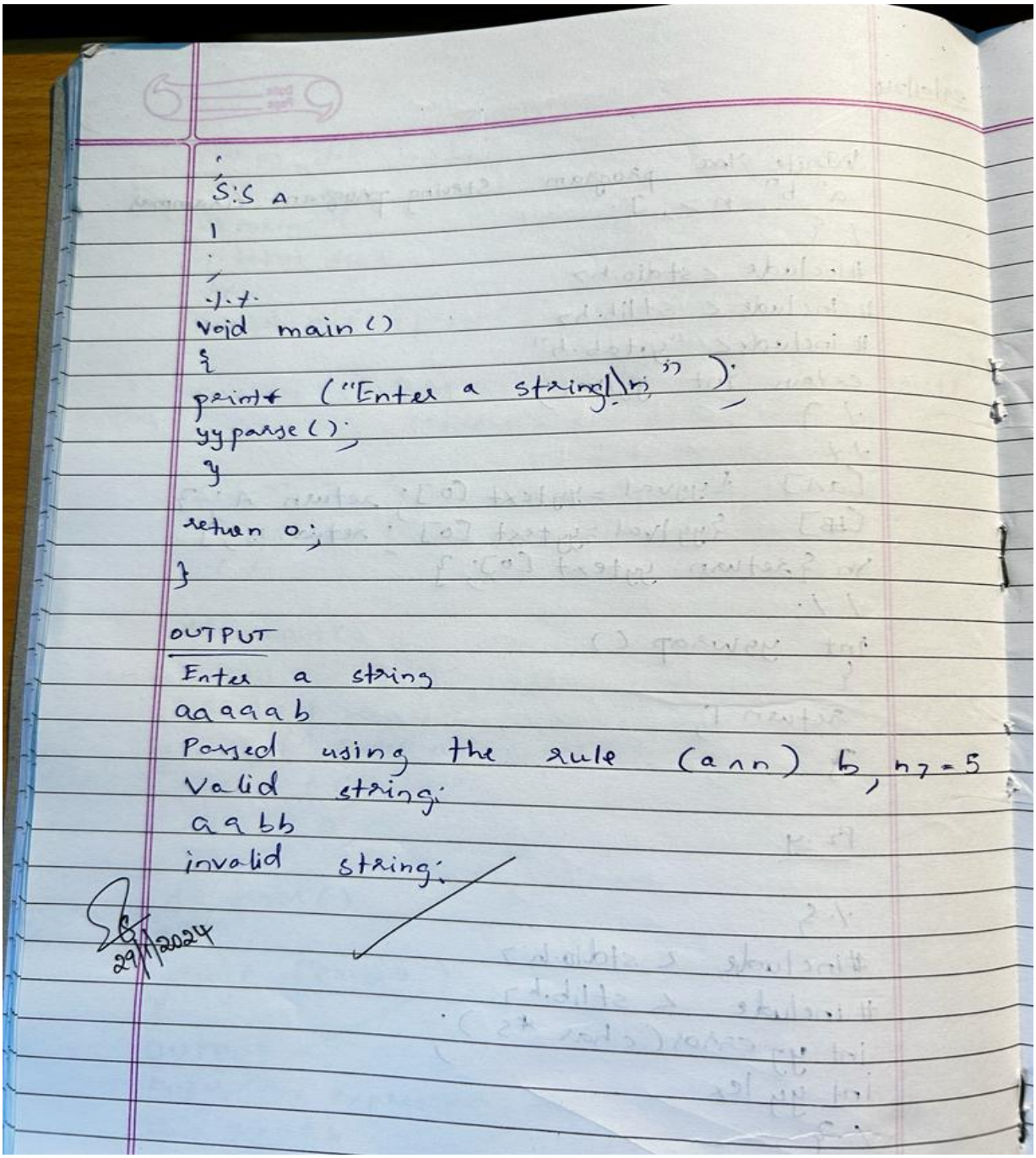
```
Enter an arithmetic expression:
2++3-
Invalid expression!
```

```
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



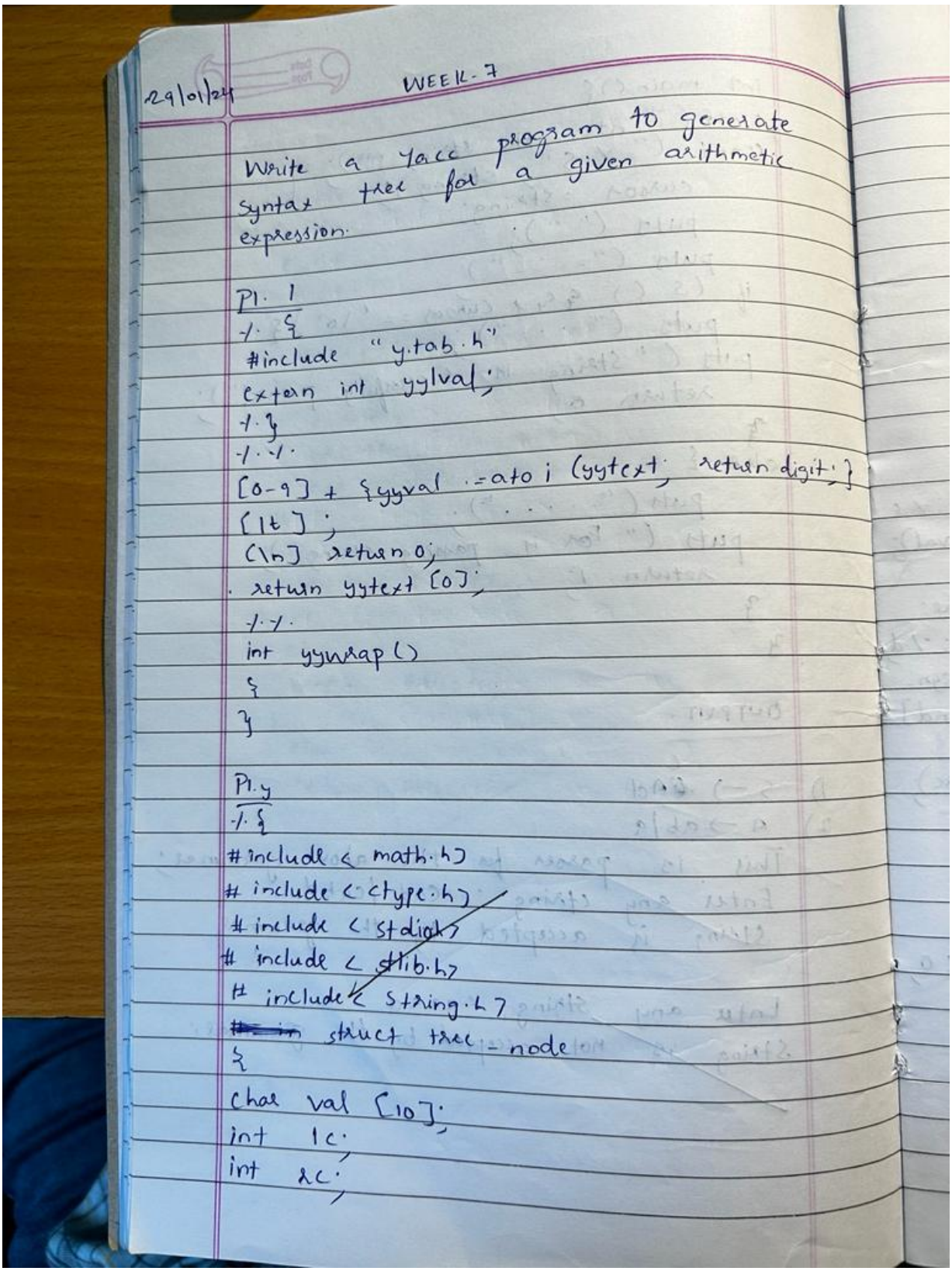


Output

```
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



29/01/24 WEEK-7

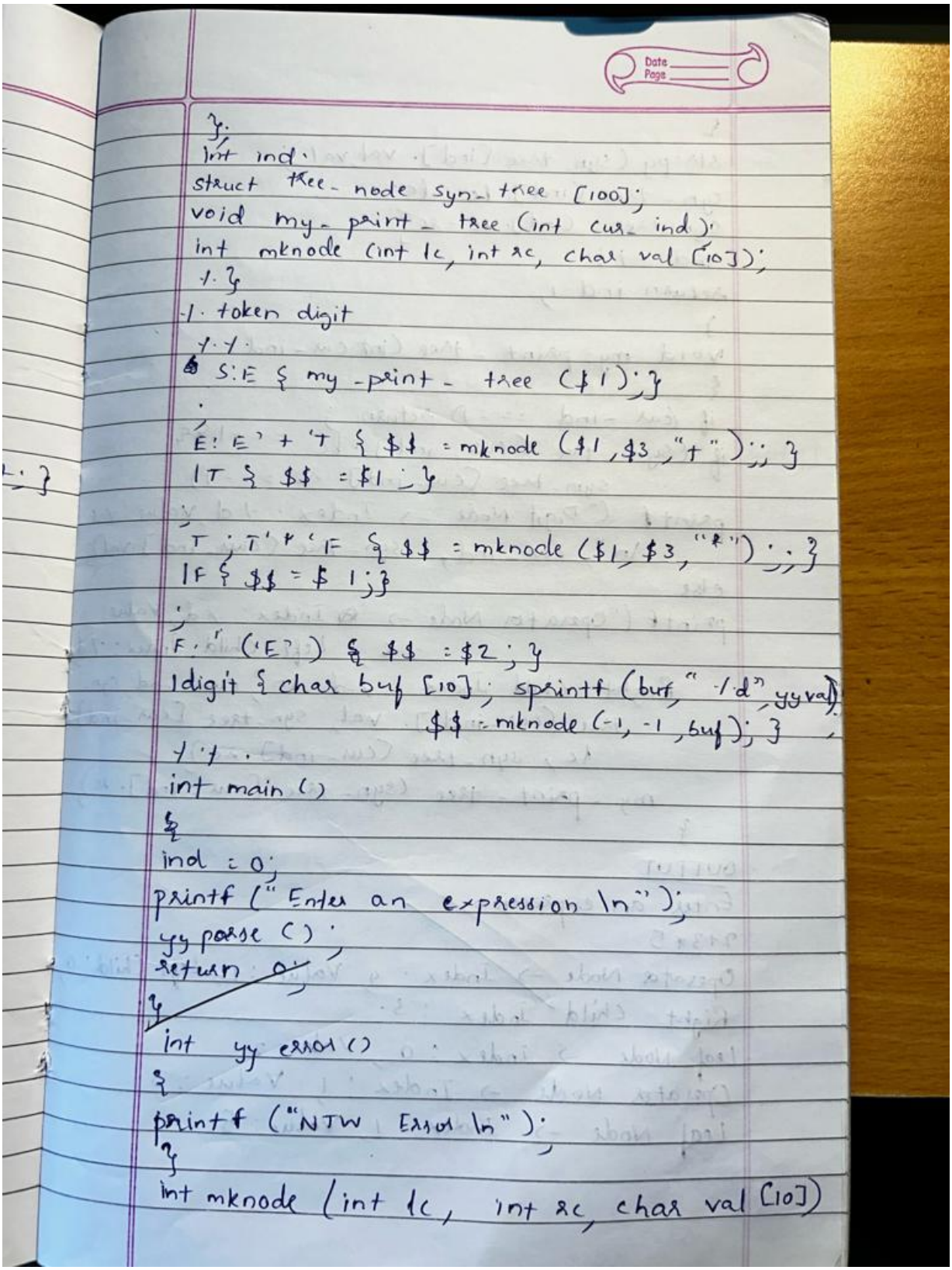
Write a yacc program to generate syntax tree for a given arithmetic expression.

Pl.l

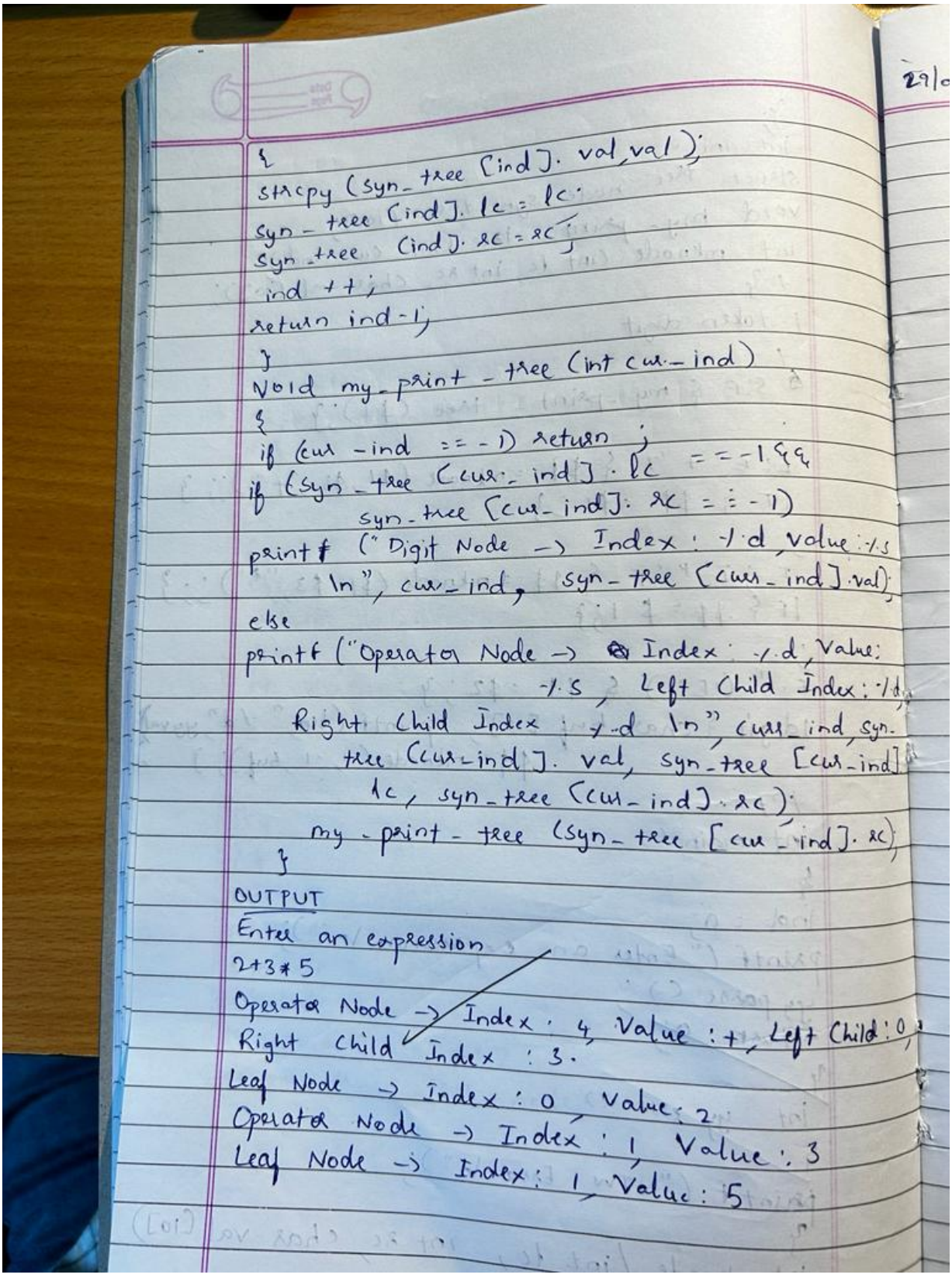
```
%{
#include "y.tab.h"
extern int yylval;
%}
%%
[0-9]+ {yyval = atoi(yytext); return digit;}
[\t];
[\n] return 0;
. return yytext[0];
%%
int yywrap()
{
}
```

Pl.y

```
%{
#include <math.h>
#include <ctype.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
struct tree_node
{
char val[10];
int lc;
int rc;
```



```
%}
int ind;
struct tree_node syn_tree [100];
void my_print_tree (int cur_ind);
int mknode (int lc, int rc, char val [10]);
%}
%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 ()
{
printf ("NTW Error \n");
}
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 my_print_tree (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, syn_tree [cur_ind]. val);
else
printf ("Operator Node -> Index : %d, Value:
    %s, Left Child Index : %d,
  Right Child Index : %d \n", cur_ind, syn.
    tree [cur_ind]. val, syn_tree [cur_ind].
    lc, syn_tree [cur_ind]. rc);
  my_print_tree (syn_tree [cur_ind]. rc);
}
```

OUTPUT

Enter an expression

2+3*5

Operator Node -> Index : 4, Value : +, Left Child : 0,
Right Child Index : 3.
Leaf Node -> Index : 0, Value : 2
Operator Node -> Index : 1, Value : 3
Leaf Node -> Index : 1, Value : 5

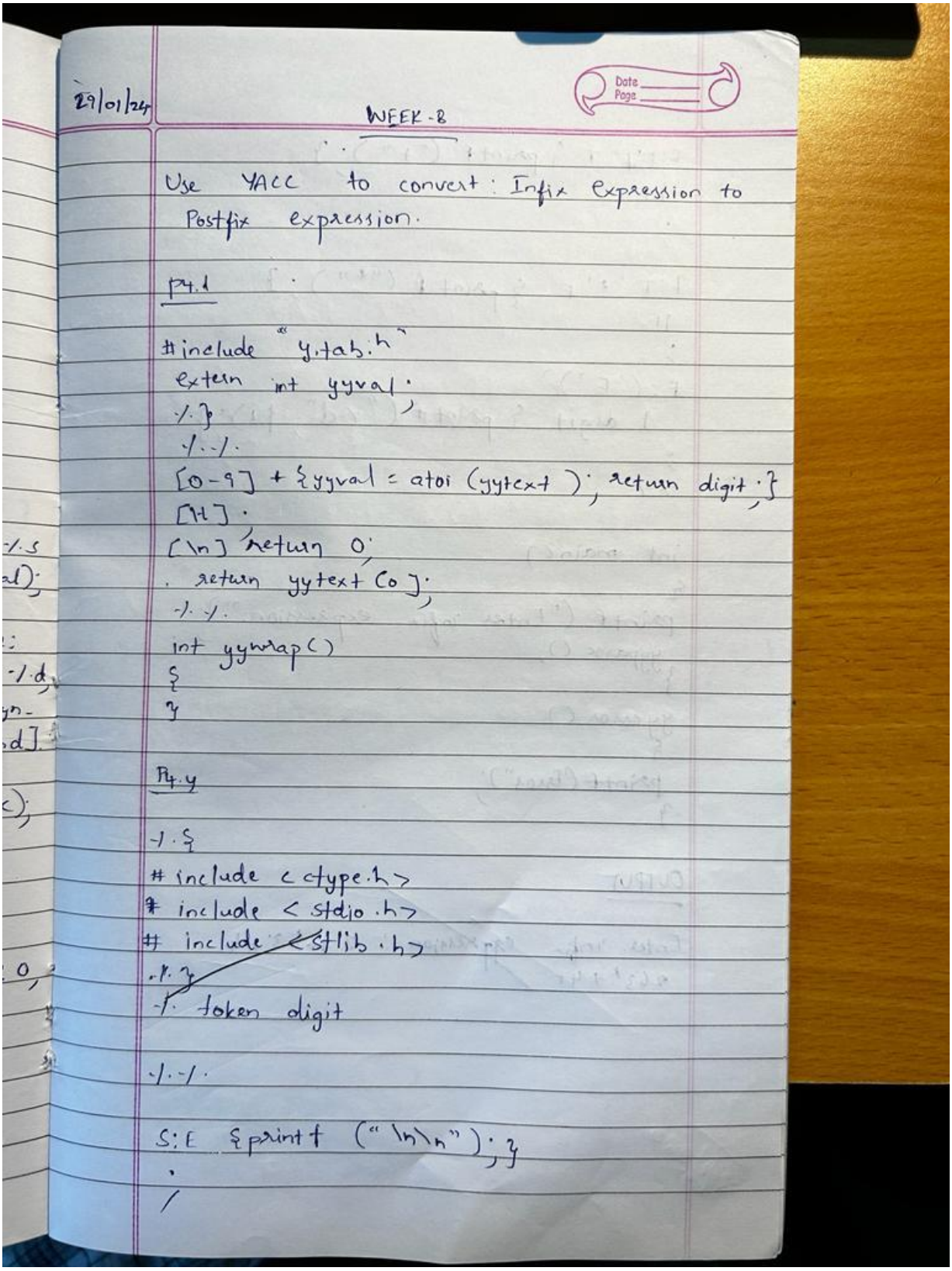
Output

```
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


29/01/24

WEEK-8

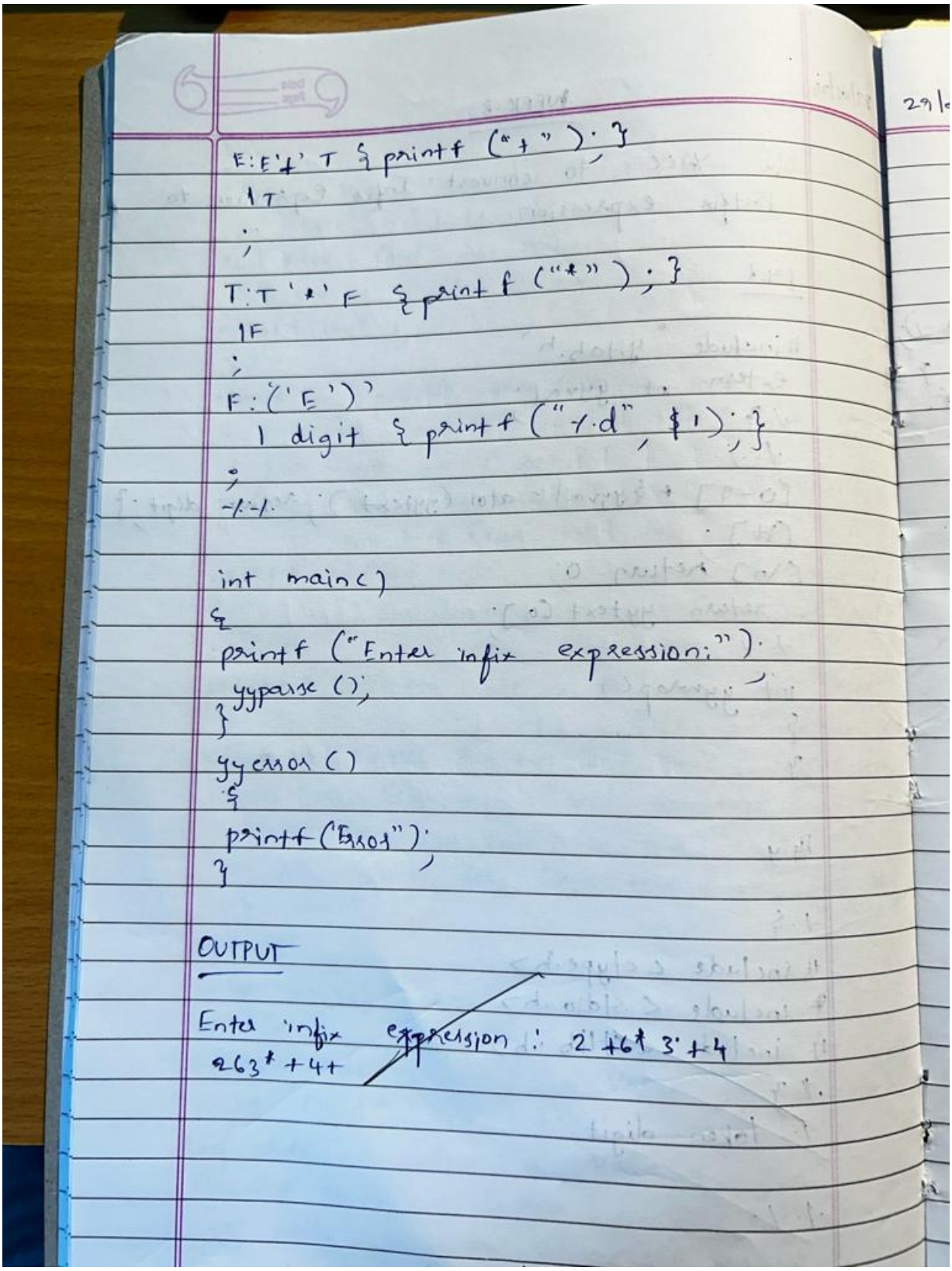
Use YACC to convert: Infix Expression to Postfix expression.

P4.l

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

P4.y

```
%{
#include <ctype.h>
#include <stdio.h>
#include <stdlib.h>
%}
%token digit

%%

S: E {printf("\n\n");}
;
/
```



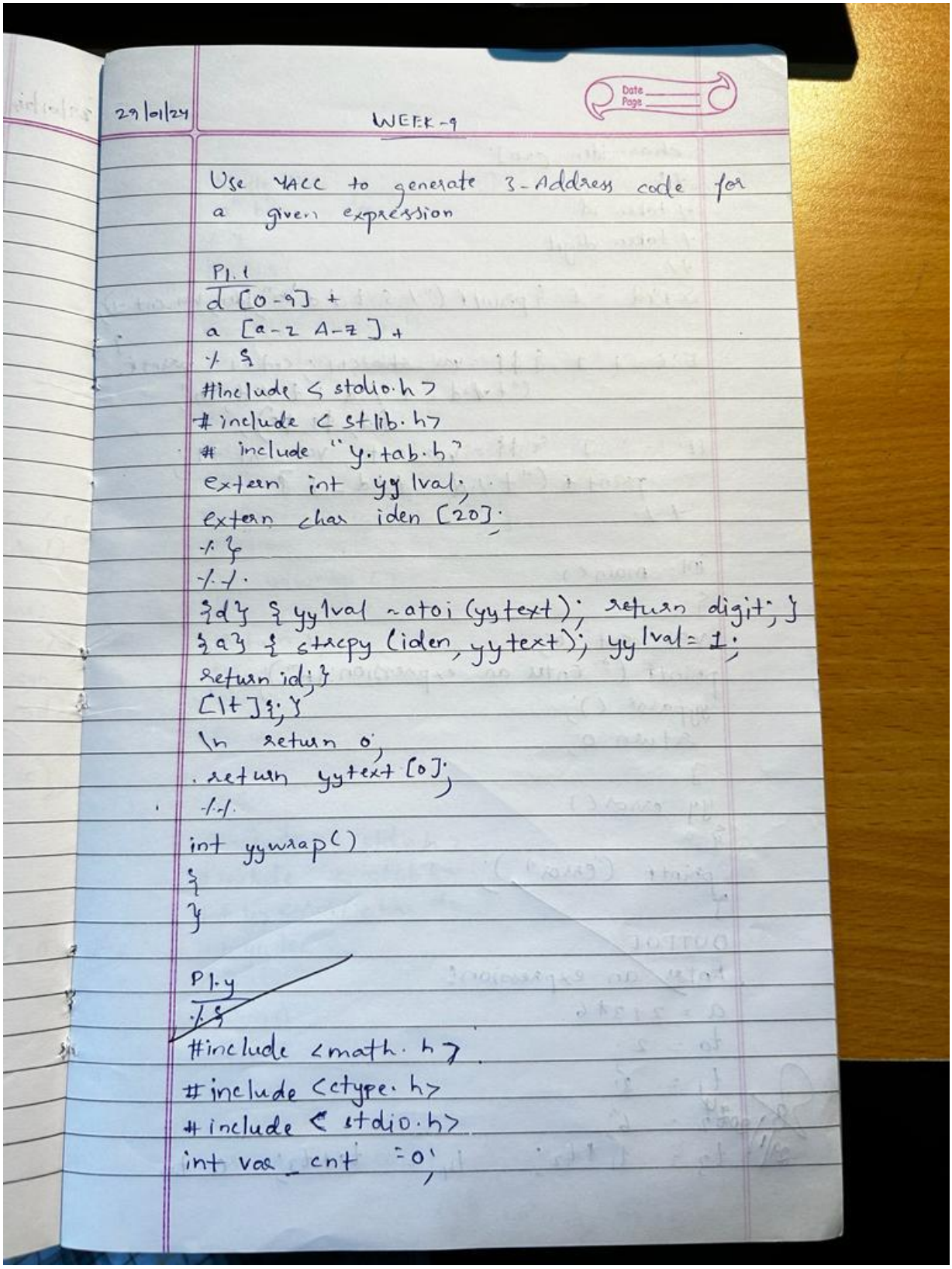
Output

```
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



29/01/24

WEEK-9

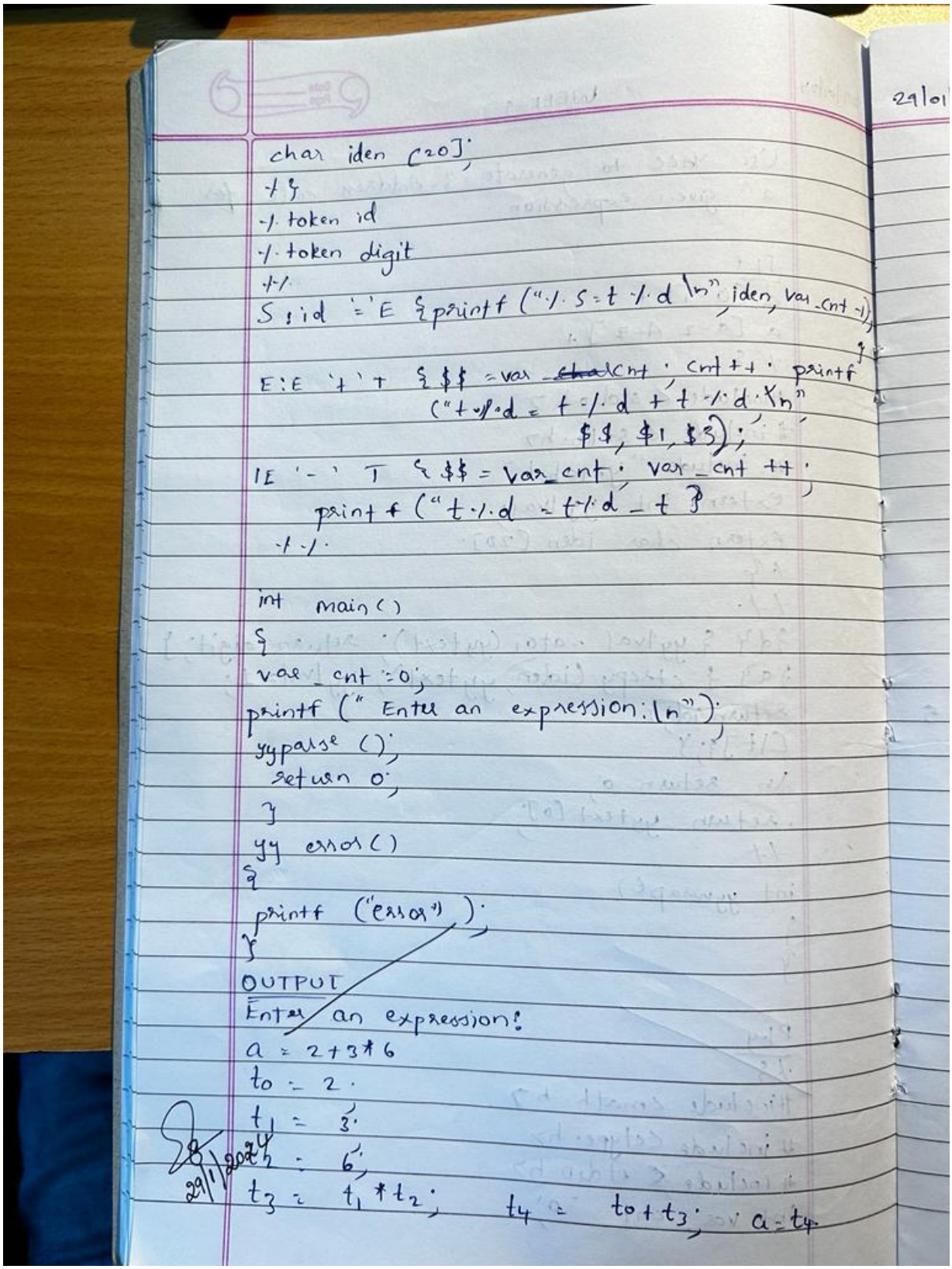
Use YACC to generate 3-Address code for a given expression

P1.l

```
d [0-9] +
a [a-z A-Z] +
%{
#include <stdio.h>
#include <stlib.h>
#include "y.tab.h"
extern int yylval;
extern char iden [20];
%}
%%
{d} { yylval = atoi (yytext); return digit; }
{a} { strcpy (iden, yytext); yylval = 1;
return id; }
[\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 ("%d S = t%d \n", iden, var_cnt-1);
              }
E : E '+' T { $$ = var_cnt; cnt++; printf
              ("t%d = t%d + t%d \n",
                $$, $1, $3);
IE '-' T { $$ = var_cnt; var_cnt ++;
    printf ("t%d = t%d - t%d
%%
int main()
{
var_cnt = 0;
printf (" Enter an expression: \n");
yyparse ();
 return 0;
 }
yy error()
{
printf ("error");
}
```

OUTPUT

Enter an expression:

a = 2+3*6

$t_0 = 2$;

$t_1 = 3$;

$t_2 = 6$;

$t_3 = t_1 * t_2$; $t_4 = t_0 + t_3$; $a = t_4$

29/1/2024

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