VISVESVARAYA TECHNOLOGICAL UNIVERSITY

"JnanaSangama", Belgaum -590014, Karnataka.

LAB REPORT

on

COMPILER DESIGN

Submitted by

Gunjal Kothari (1BM21CS274)

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 **Gunjal Kothari (1BM21CS274)** , 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, Gunjal Kothari (1BM21CS274), student of 5th Semester, B.E, Department ofComputer 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 ofEngineering, Bangalore during the academic semester November-2023-February-2024.

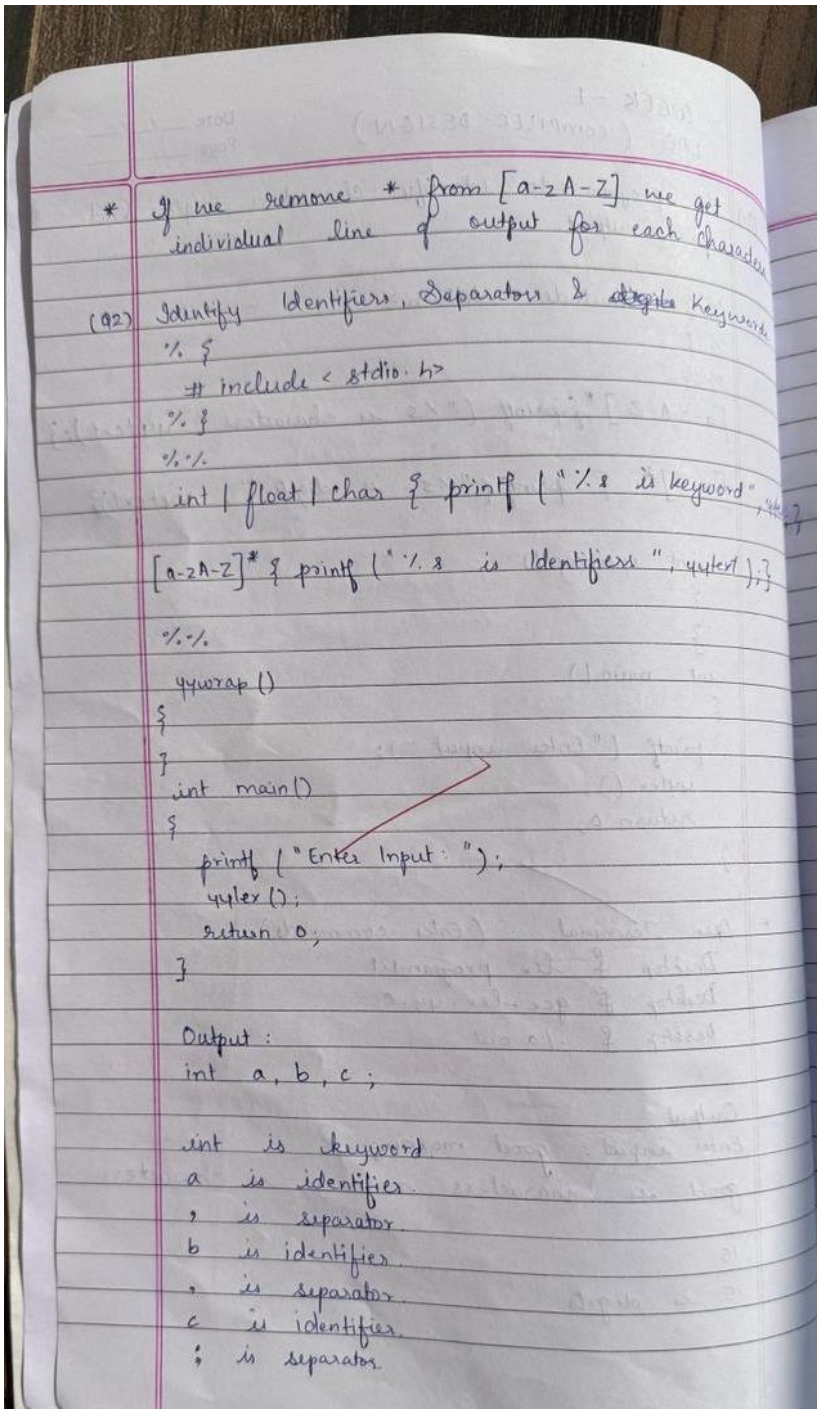
I also declare that to the best of my knowledge and belief, the development reported here is not from part of any other report by any other students.

TABLE OF CONTENTS

Lab 1

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

Code:



* If we remove * from [a-zA-Z] we get individual line of output for each character

(Q2) Identify Identifiers, Separators & ~~digits~~ Keywords

```
%{
  #include <stdio.h>
%}
%%
int | float | char { printf ("%s is keyword", yytext);}
[a-zA-Z]* { printf ("%s is Identifiers", yytext);}
%%
yywrap()
{
}
int main()
{
  printf ("Enter Input: ");
  yylex();
  return 0;
}
```

Output:

```
int a, b, c;
```

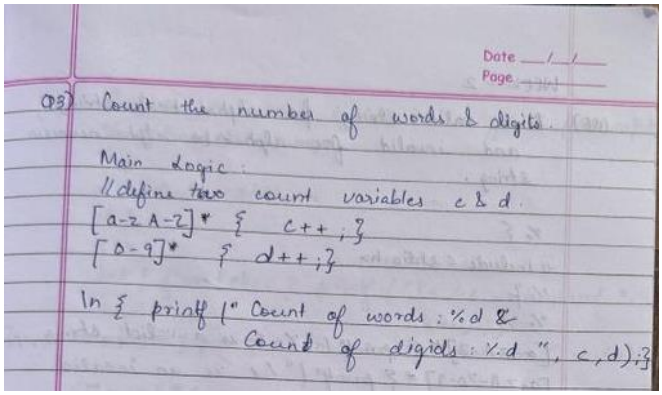
int is keyword
a is identifier
, is separator
b is identifier
, is separator
c is identifier
; is separator

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 characters and digits in a string.

Code



Date __/__/__
Page

Q5) Count the number of words & digits.

Main Logic:
// define two count variables c & d.
[a-z A-Z]* { c++; }
[0-9]* { d++; }

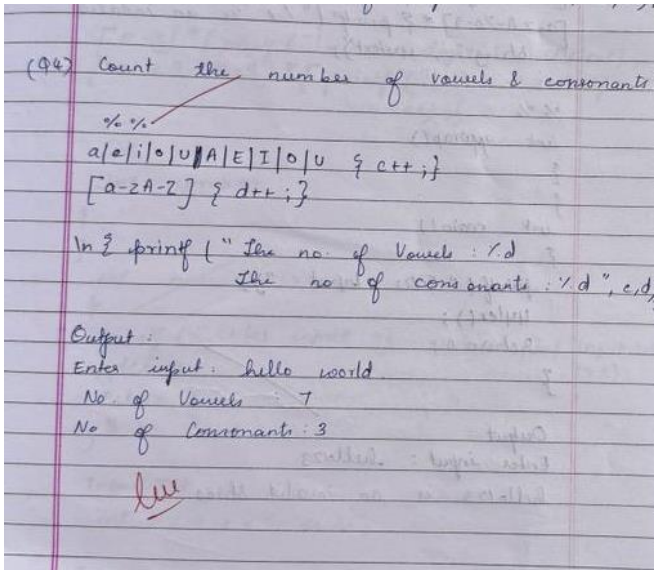
In { printf ("Count of words : %d &
Count of digits : %d ", c, d); }

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



```
(Q4) Count the number of vowels & consonants
%%
a|e|i|o|u|A|E|I|O|U { c++;}
[a-zA-Z] { d++;}
\n { printf("The no. of Vowels : %d
        The no. of consonants : %d", c,d);
Output:
Enter input: hello world
No. of Vowels : 7
No. of Consonants : 3
```

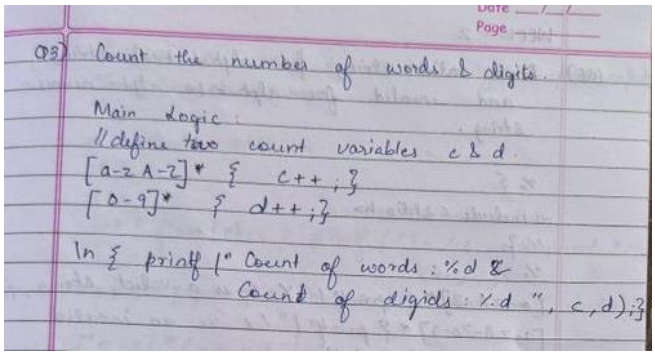
Output

```
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
^C
```

Lab 2

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

Code

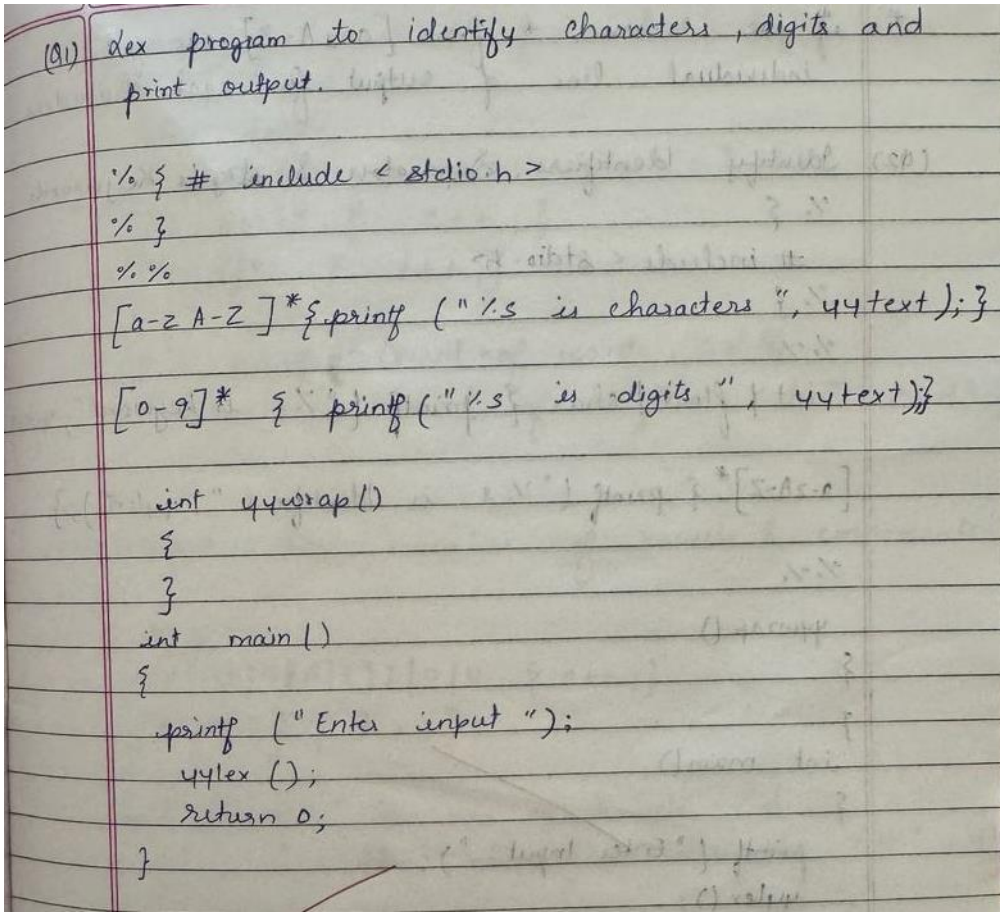


Output

```
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



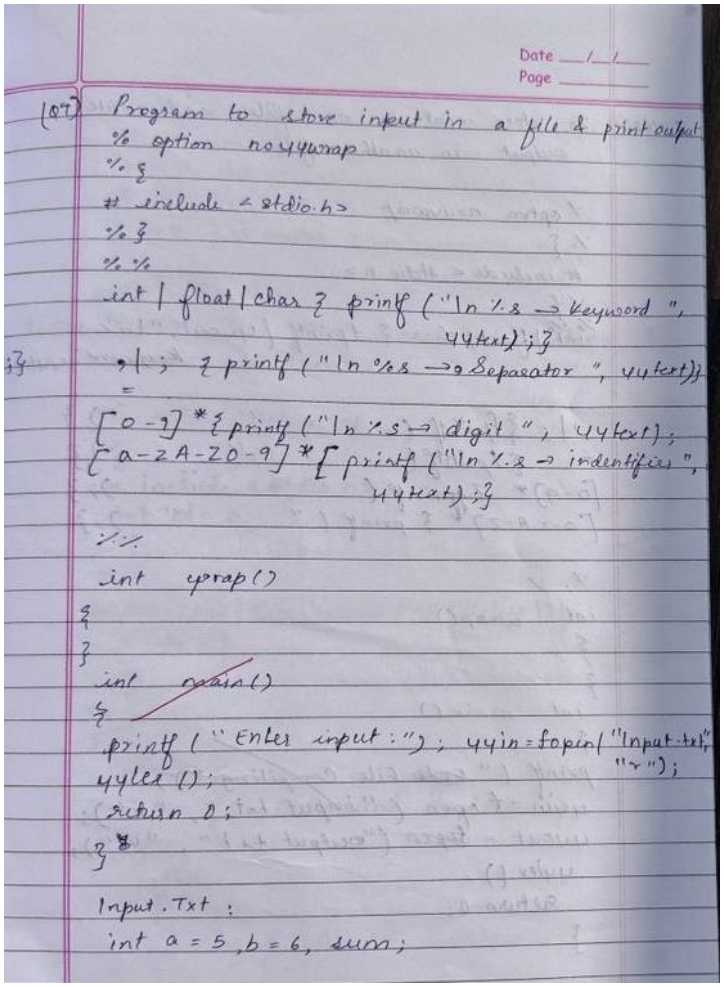
```
(Q1) Lex program to identify characters, digits and
print output.

%{ # include <stdio.h>
%}
%%
[a-z A-Z]*{printf("%s is characters", yytext);}

[0-9]* { printf("%s is digits", yytext);}

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

Output

```
Enter a string:
HelloWorld
Characters

1234
Digits
Hello123
Invalid input!
```

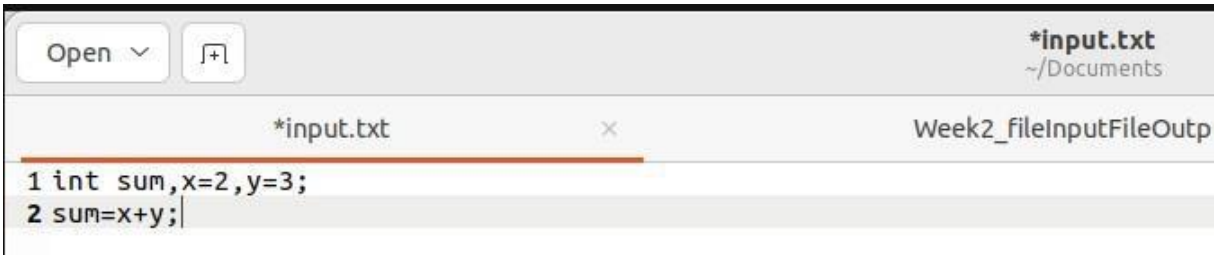
2.3 Write a program in lex to identify tokens in a program by taking input from a file and printing the output on the terminal.

Code



```
Date __/__/____
Page _____

(07) Program to store input in a file & print output
%option noyywrap
%{
#include <stdio.h>
%}
%%
int | float | char { printf("\n %s -> Keyword ", yytext); }
,|; { printf("\n %s -> Separator ", yytext); }
=
[0-9]* { printf("\n %s -> digit ", yytext); }
[a-zA-Z0-9]* { printf("\n %s -> indentifier ", yytext); }
%%
int yywrap()
{
}
int main()
{
printf("Enter input:"); yyin=fopen("Input.txt", "r");
yylex();
return 0;
}

Input.Txt :
int a = 5, b = 6, sum;
```

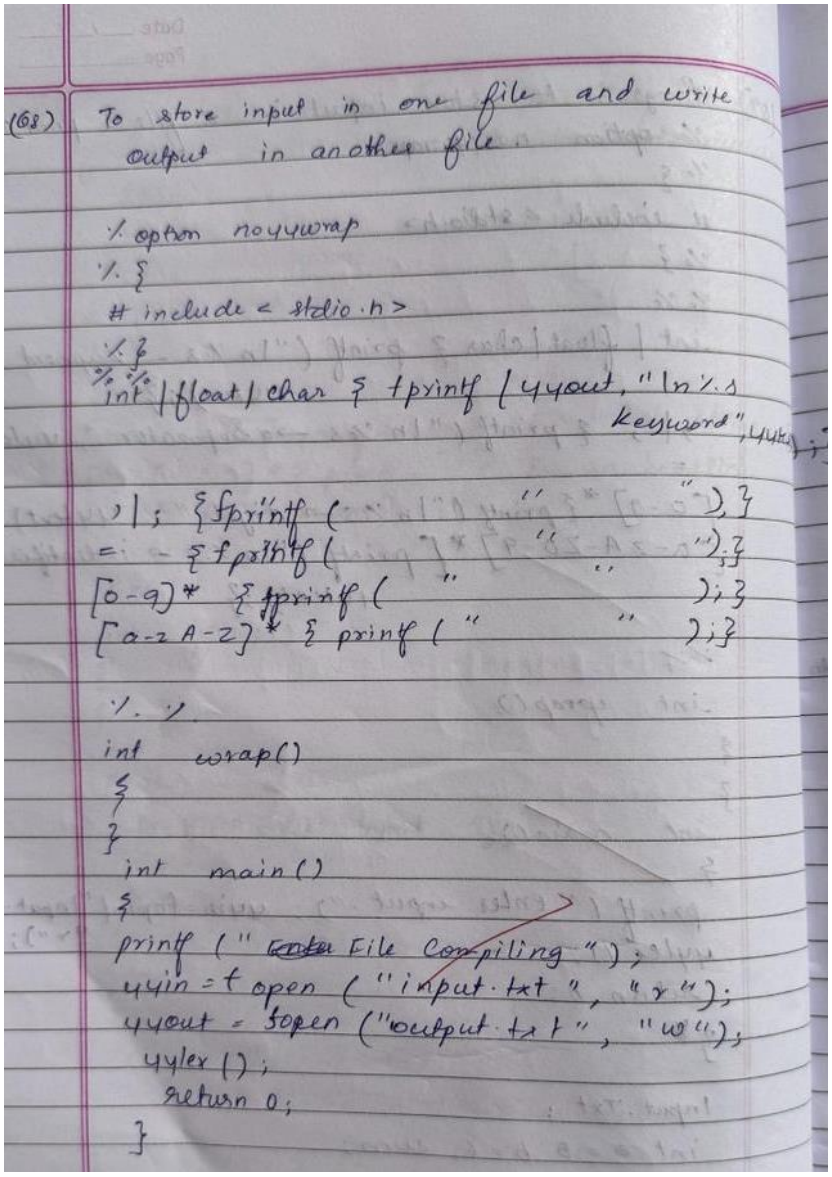
Output




```
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.

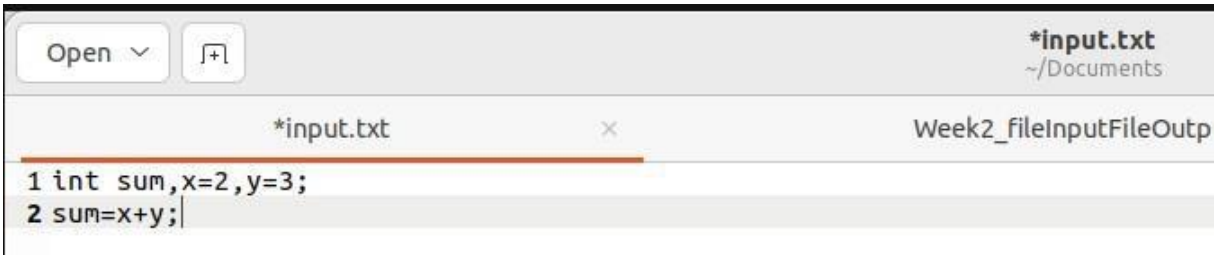
Code



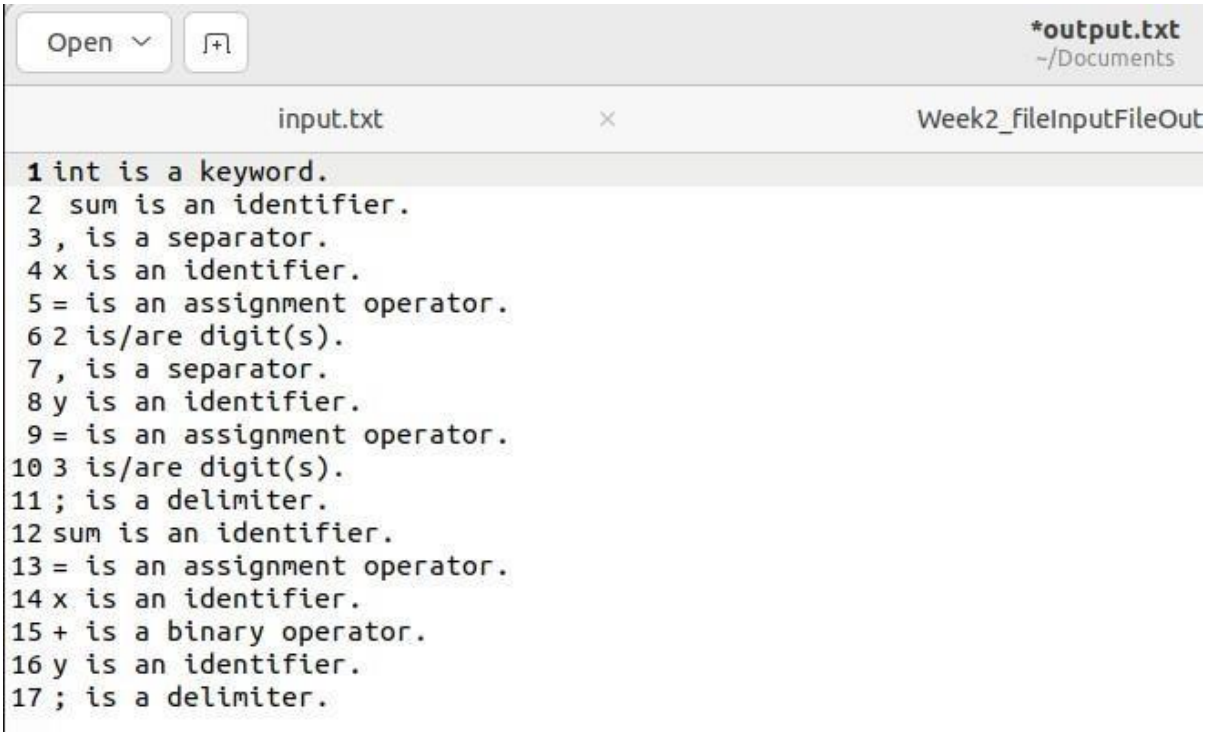
(68) To store input in one file and write output in another file

```
%option noyywrap
%{
#include <stdio.h>
%}
%%
int|float|char { fprintf(yyout,"\n%s keyword",yytext); }
);|;{fprintf(" ");}
={fprintf(" ");}
[0-9]* {fprintf(" ");}
[a-zA-Z]* {printf(" ");}
%%
int wrap()
{
}
int main()
{
printf("File Compiling");
yyin=fopen("input.txt","r");
yyout=fopen("output.txt","w");
yylex();
return 0;
}
```

Output

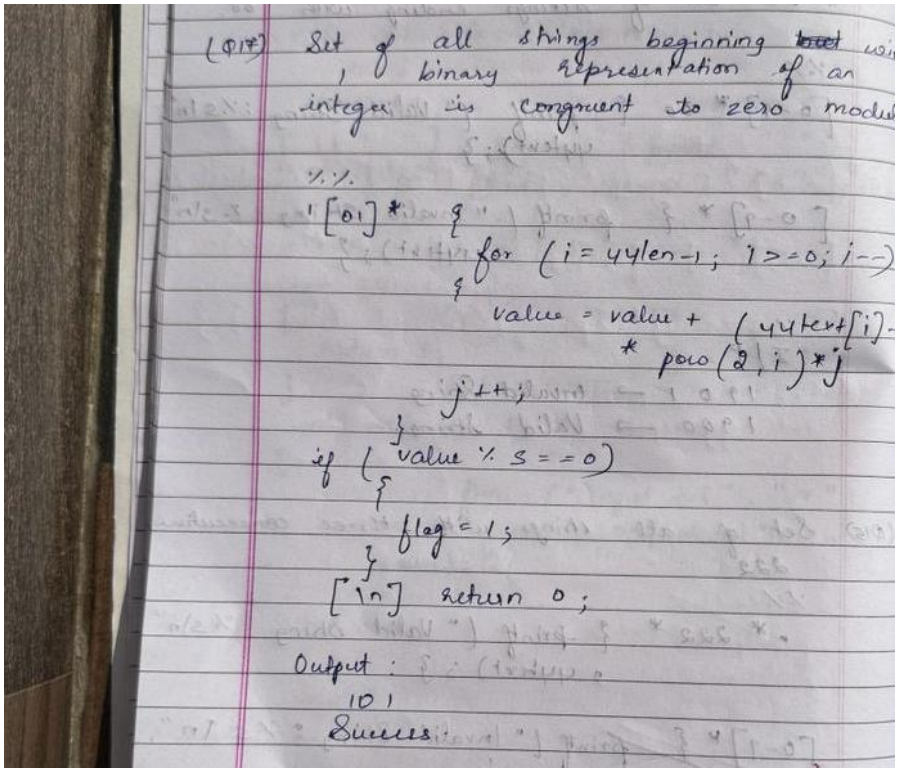



```
Printed in output.txt
```



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

Code



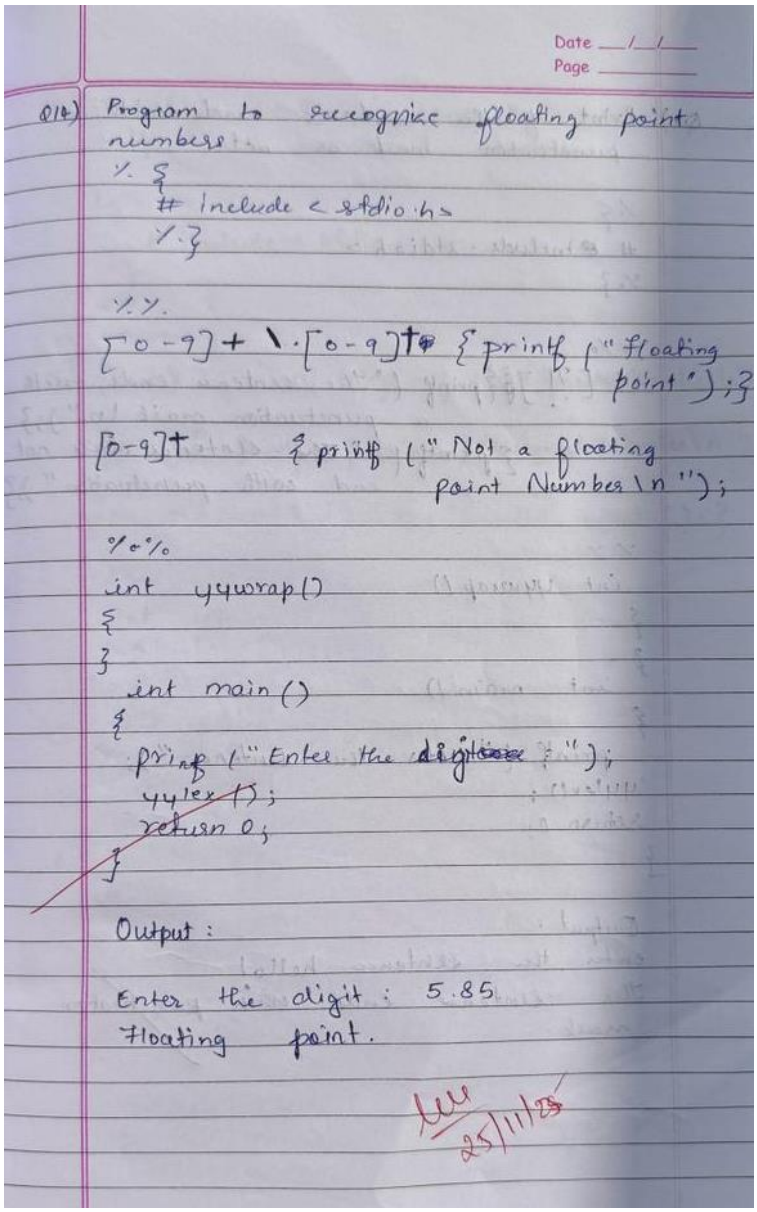
Output

```
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



Q1) Program to recognize floating point numbers

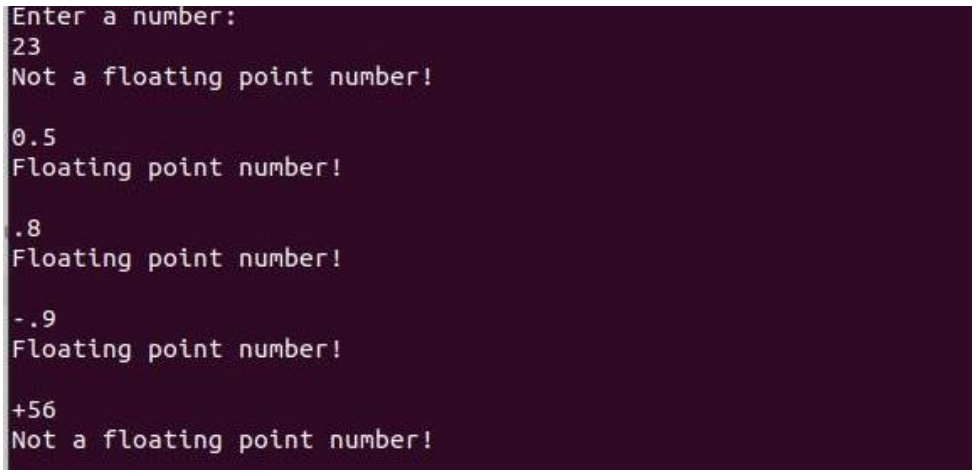
```
%{
#include <stdio.h>
%}

%%
[0-9]+\.[0-9]+ {printf("floating point");}

[0-9]+ {printf("Not a floating point Numbers\n");}

%%
int yywrap()
{
}
int main()
{
printf("Enter the digits : ");
yylex();
return 0;
}
```

Output :

```
Enter the digit : 5.85
Floating point.
```

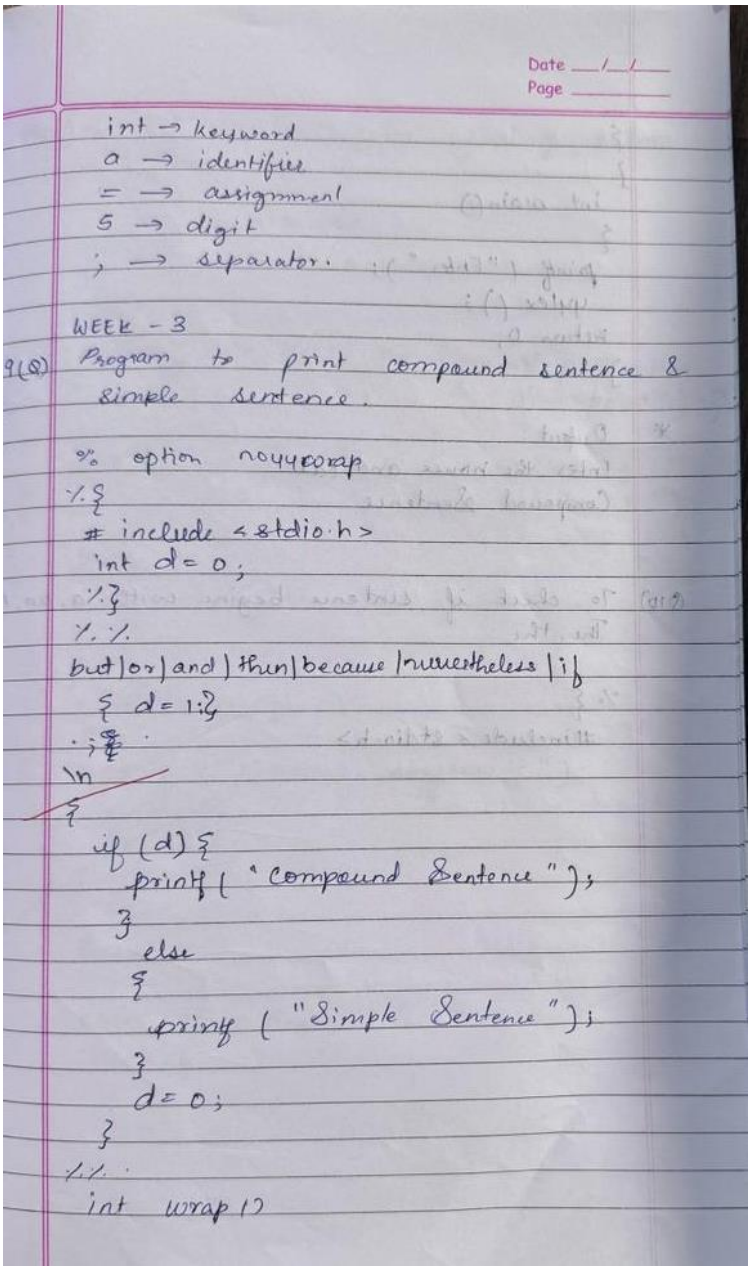
25/11/25

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



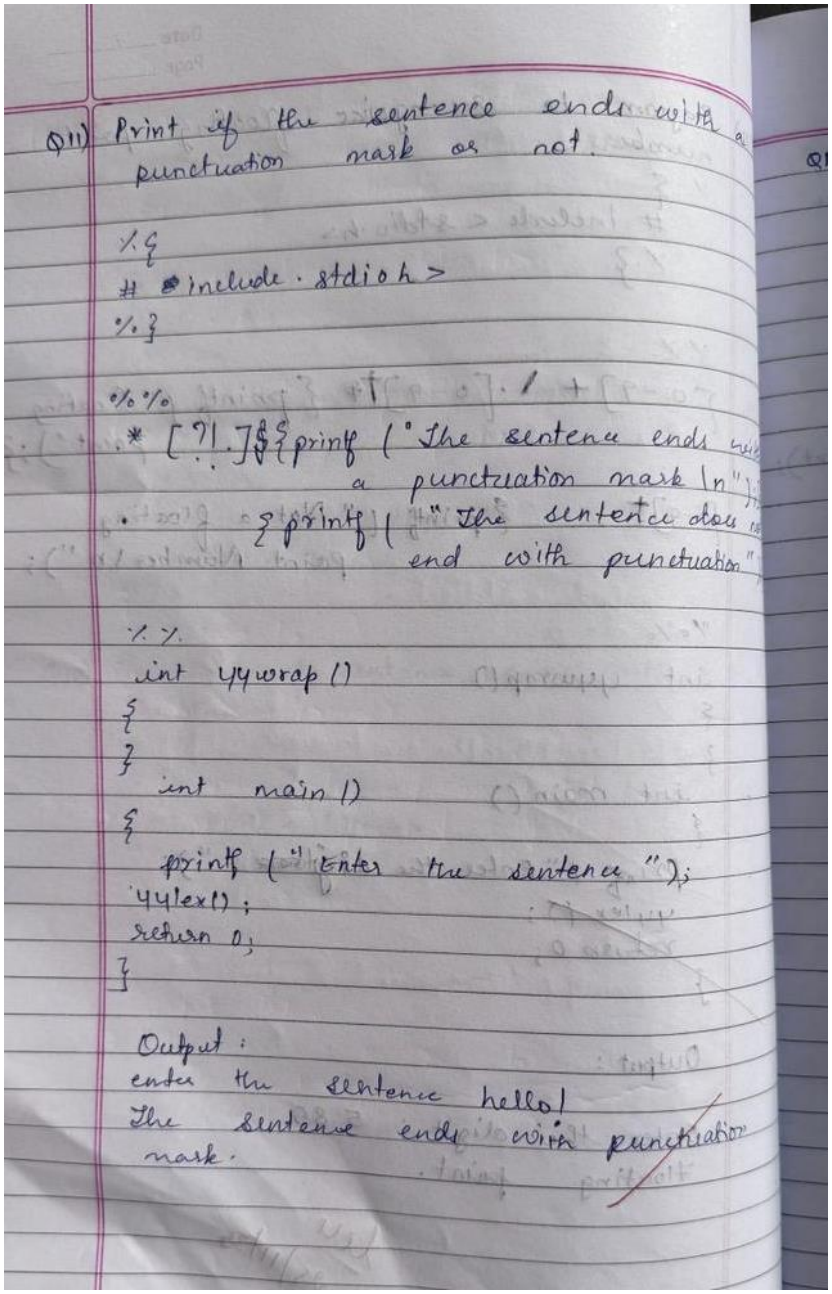
```
Date __/__/__
Page _____

int → keyword
a → identifier
= → assignment
5 → digit
; → separator.

WEEK - 3
Program to print compound sentence & simple sentence.

%% option noyywrap
%{
# include <stdio.h>
int d = 0;
%}
%%
but|or|and|then|because|nevertheless|if
  { d = 1;}
.;
\n
{
  if (d) {
    printf ("Compound Sentence");
  }
    else
    {
      printf ("Simple Sentence");
    }
    d = 0;
  }
%%
int wrap()
```

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



Q11) Print if the sentence ends with a punctuation mark or not.

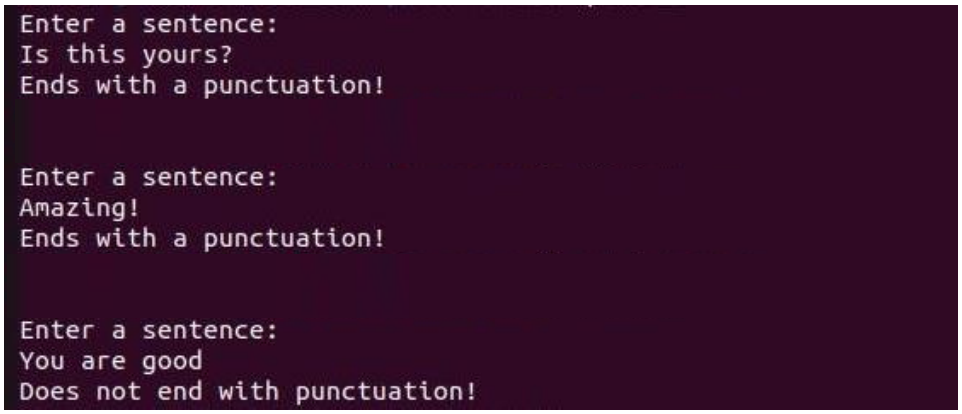
```
%{
#include<stdio.h>
%}

%%
.*[?|.|!]$ {printf("The sentence ends with a punctuation mark\n");}
.* {printf("The sentence does not end with punctuation");}

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

Output:

enter the sentence hello!

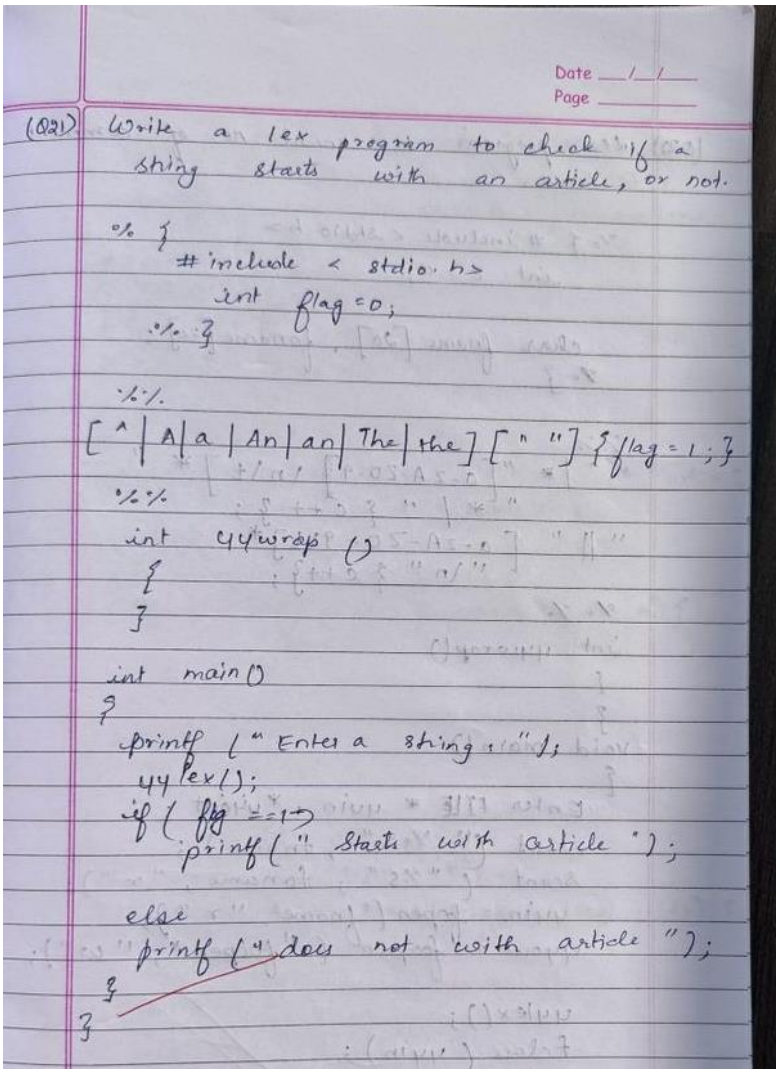
The sentence ends with punctuation mark.

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!

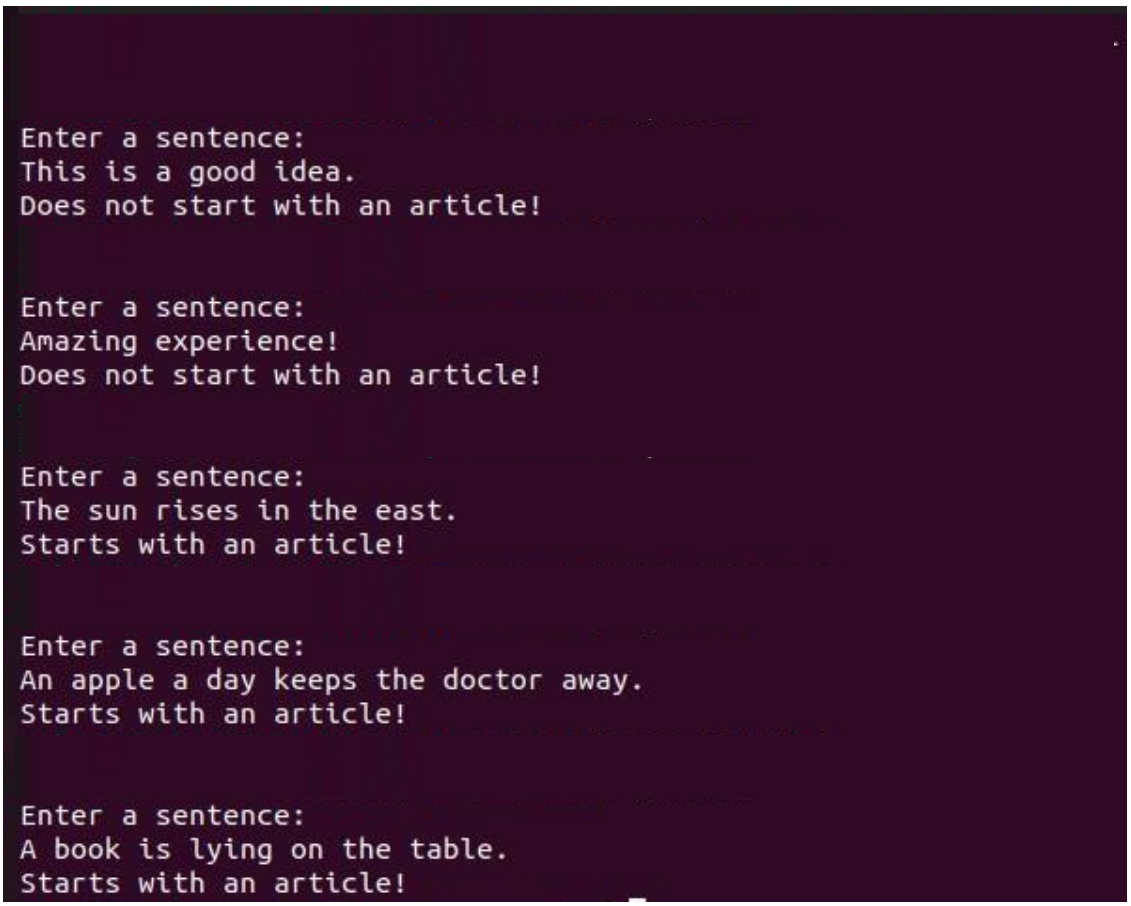
3.4 Write a program to read an input sentence and to check if the sentence begins with English articles (A, a,AN,An,THE and The).

Code



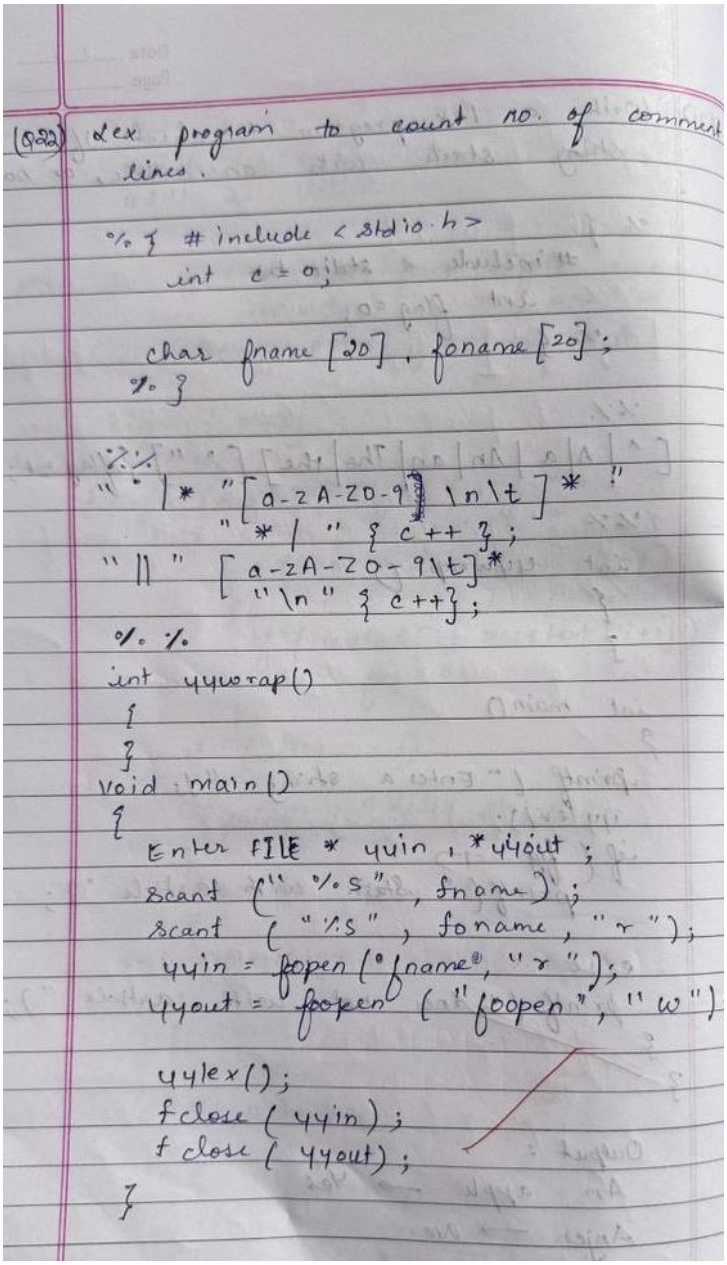
```
Date __/__/__
Page _____

(Q21) Write a lex program to check if a string starts with an article, or not.

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

%%
[^|A|a|An|an|The|the][" "]{flag=1;}
%%
int yywrap()
{
}

int main()
{
printf("Enter a string:");
yylex();
if(flag==1)
printf("Starts with article");

else
printf("does not with article");
}
```

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



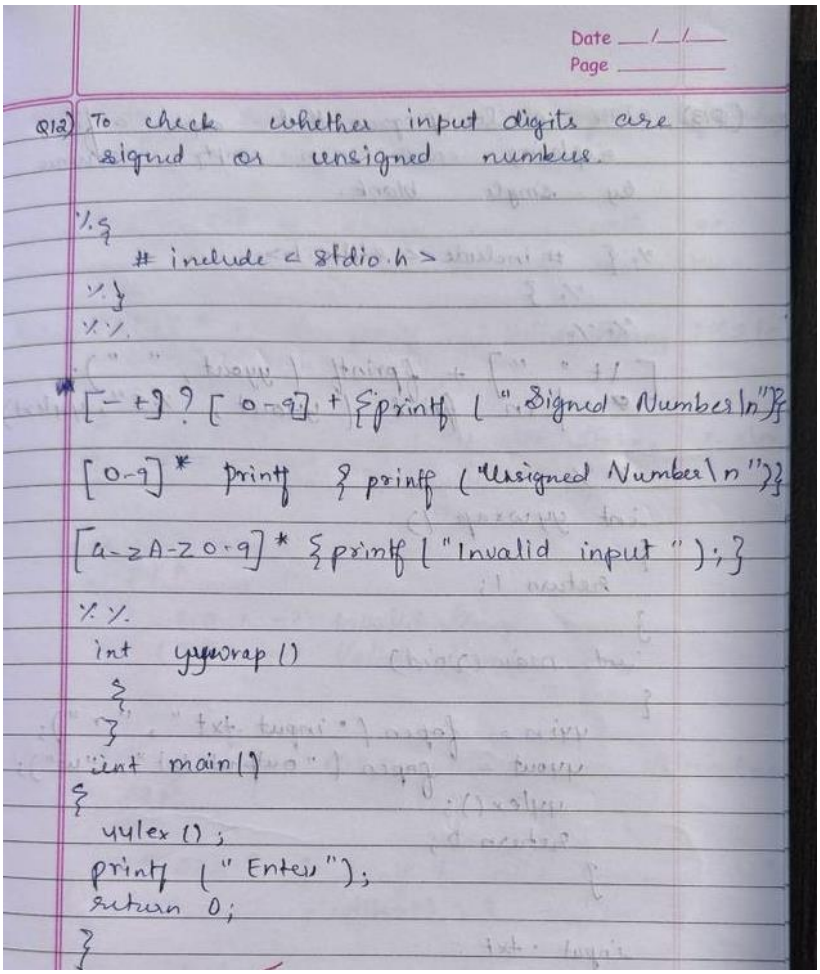
```
(3.22) Lex program to count no. of comment lines.

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

   char fname[20], foname[20];
%}

%%
"/*"[a-zA-Z0-9 \n\t]*"*/" { c++ };
"//"[a-zA-Z0-9\t]*"\n" { c++};
%%
int yywrap()
{
}
void main()
{
  Enter FILE * yyin, *yyout;
  scanf("%s", fname);
  scanf("%s", fname, "r");
  yyin = fopen(fname, "r");
  yyout = fopen("foopen", "w")

  yylex();
  fclose(yyin);
  fclose(yyout);
}
```

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



```
Q12) To check whether input digits are signed or unsigned numbers.

%{
  # include <stdio.h>
%}
%%
[-+]?[0-9]+ {printf("Signed Number\n");}
[0-9]* printf {printf("Unsigned Number\n");}
[a-zA-Z0-9]* {printf("Invalid input");}
%%
int yywrap()
{
}
int main()
{
  yylex();
printf("Enter");
return 0;
}
```

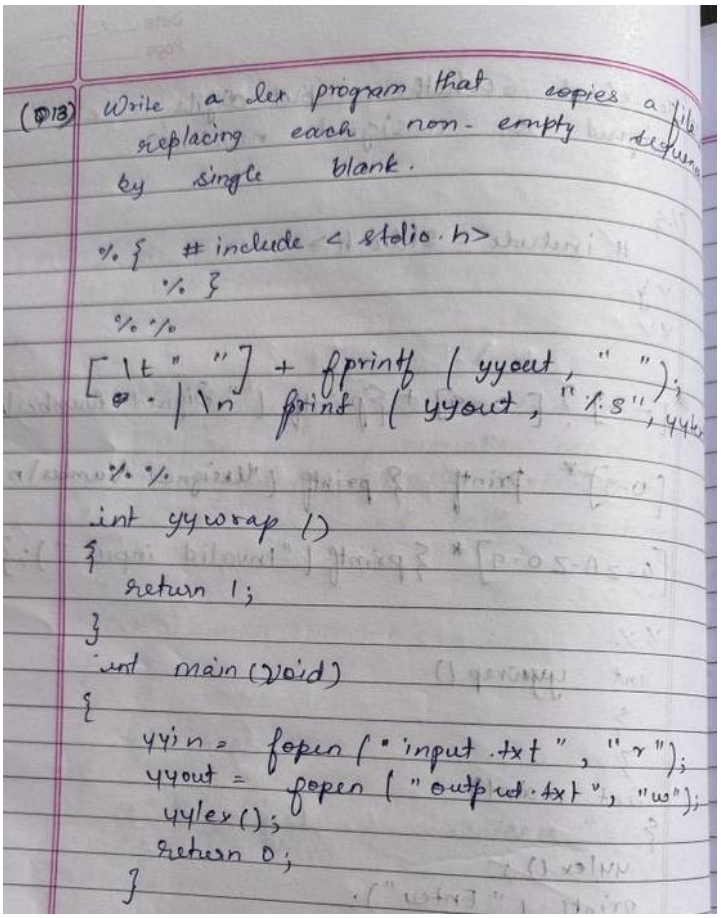
Output

```
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



```
(Q18) Write a lex program that copies a file
replacing each non-empty sequence
by single blank.

%{ #include <stdio.h>
%}
%%
[\t" "]+ fprintf(yyout, " ");
.|\n fprint(yyout, "%s", yytext);
%%
int yywrap()
{
  return 1;
}
int main(void)
{
  yyin = fopen("input.txt", "r");
  yyout = fopen("output.txt", "w");
  yylex();
  return 0;
}
```

Output


*text.txt
1 Hello World
2 Welcome to programming

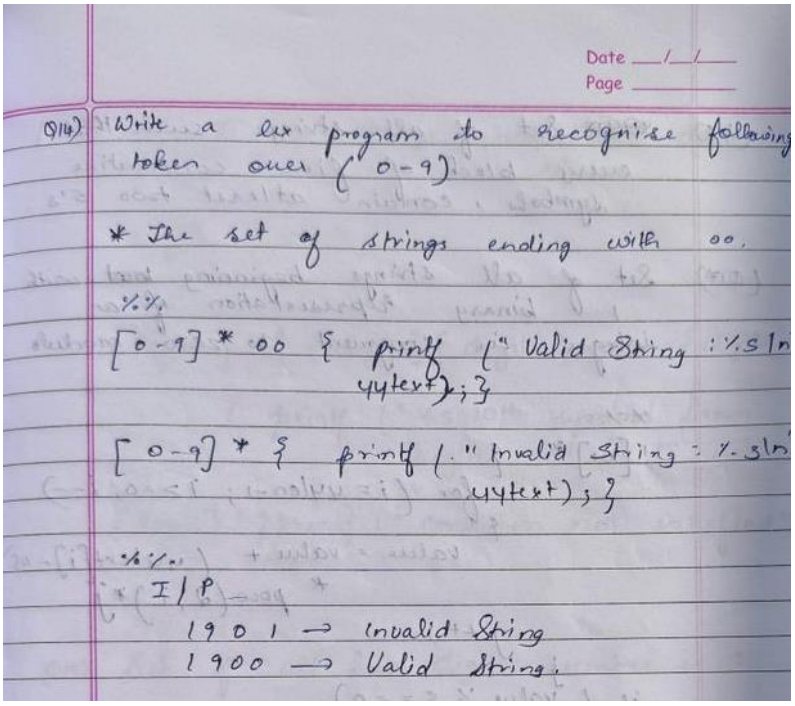

Printed!


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



Date __/__/__
Page _____

Q14) Write a lex program to recognise following token over ('0-9')

* The set of strings ending with 00.

```
%%
[0-9]* 00 { printf ("Valid String : %s\n
            yytext); }

[0-9]* { printf ("Invalid String : %s\n
         yytext); }

%%
```

I/P

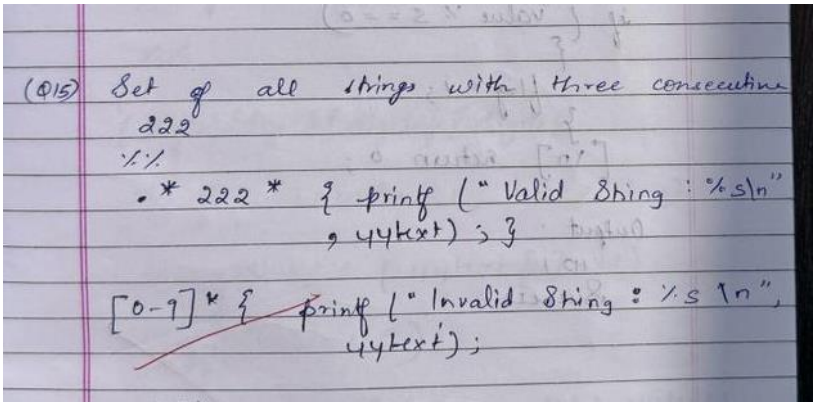
1901 → Invalid String
1900 → Valid String.

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

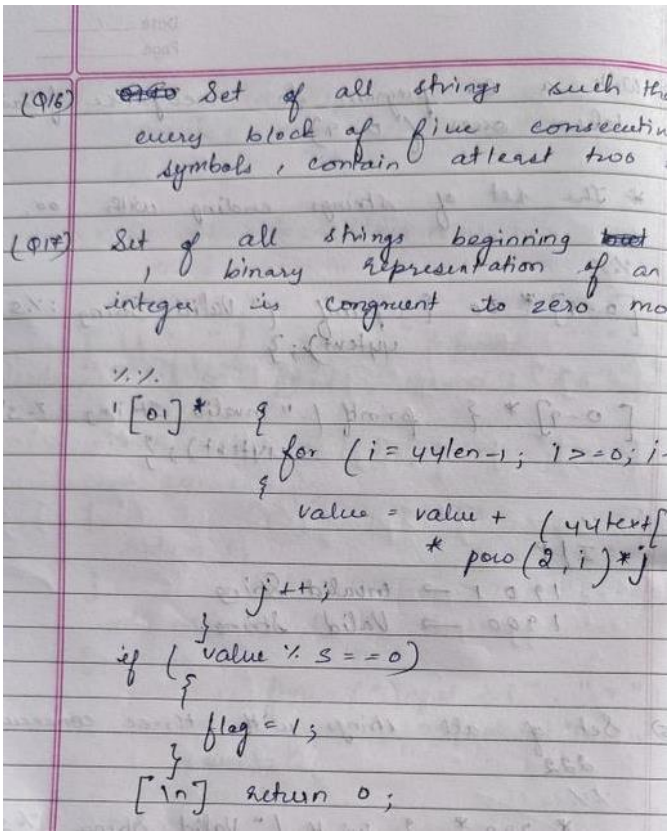


Output

```
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

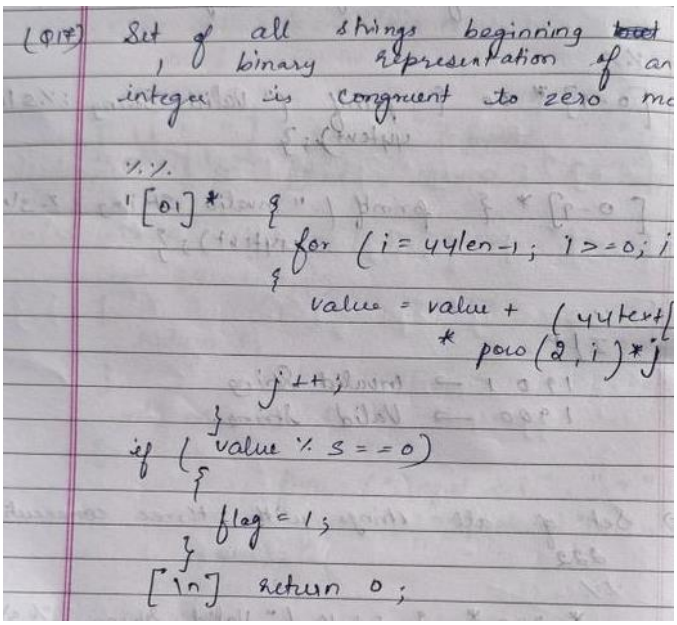


Output

```
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

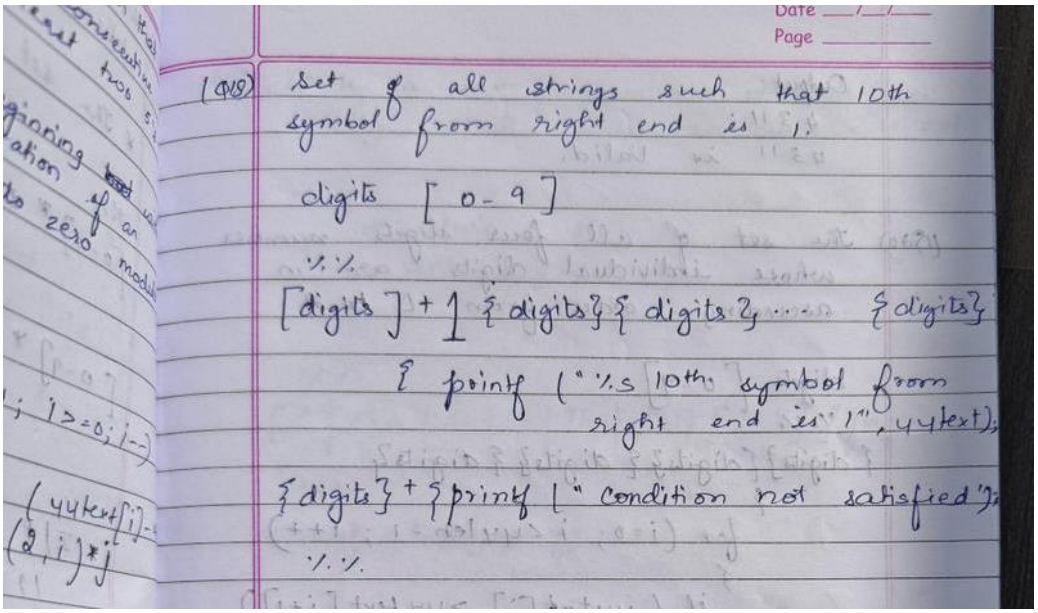


Output

```
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



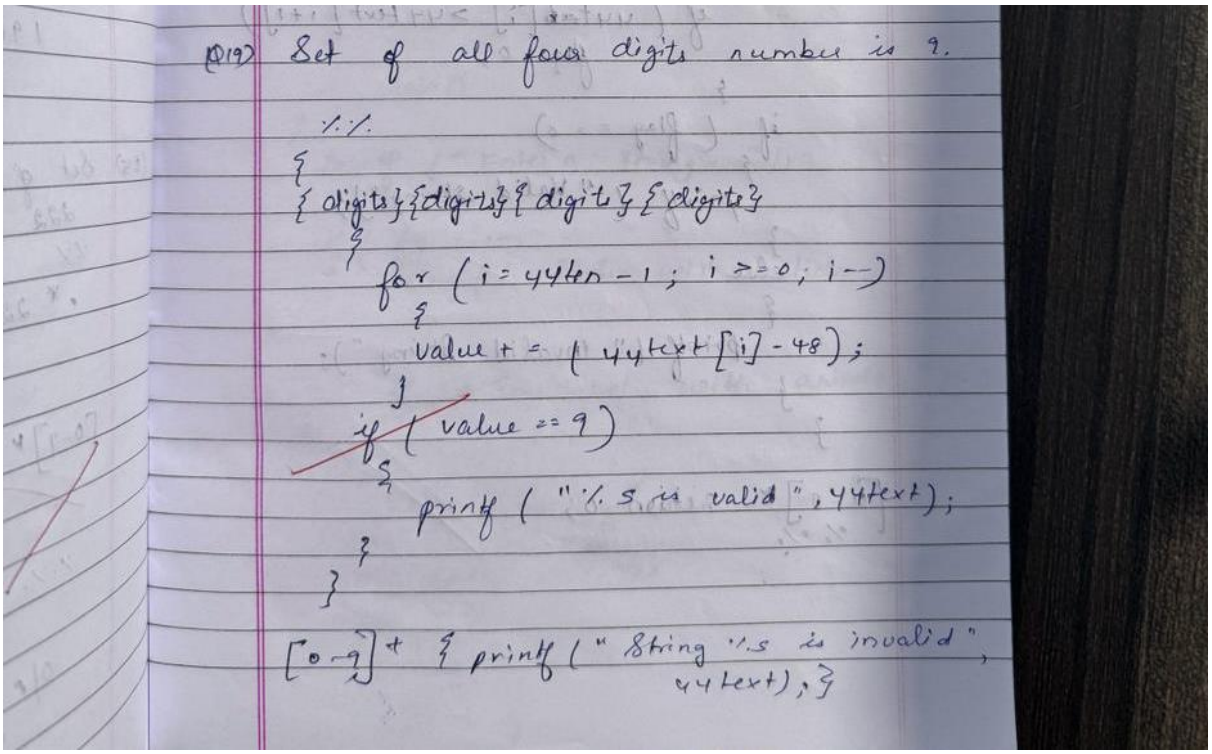
```
(Q8) Set of all strings such that 10th symbol from right end is 1.

digits [0-9]

%%
[digits]+1{digits}{digits}.....{digits}
{ printf("%s 10th symbol from right end is 1",yytext);
{digits}+{printf("condition not satisfied");
%%
```

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



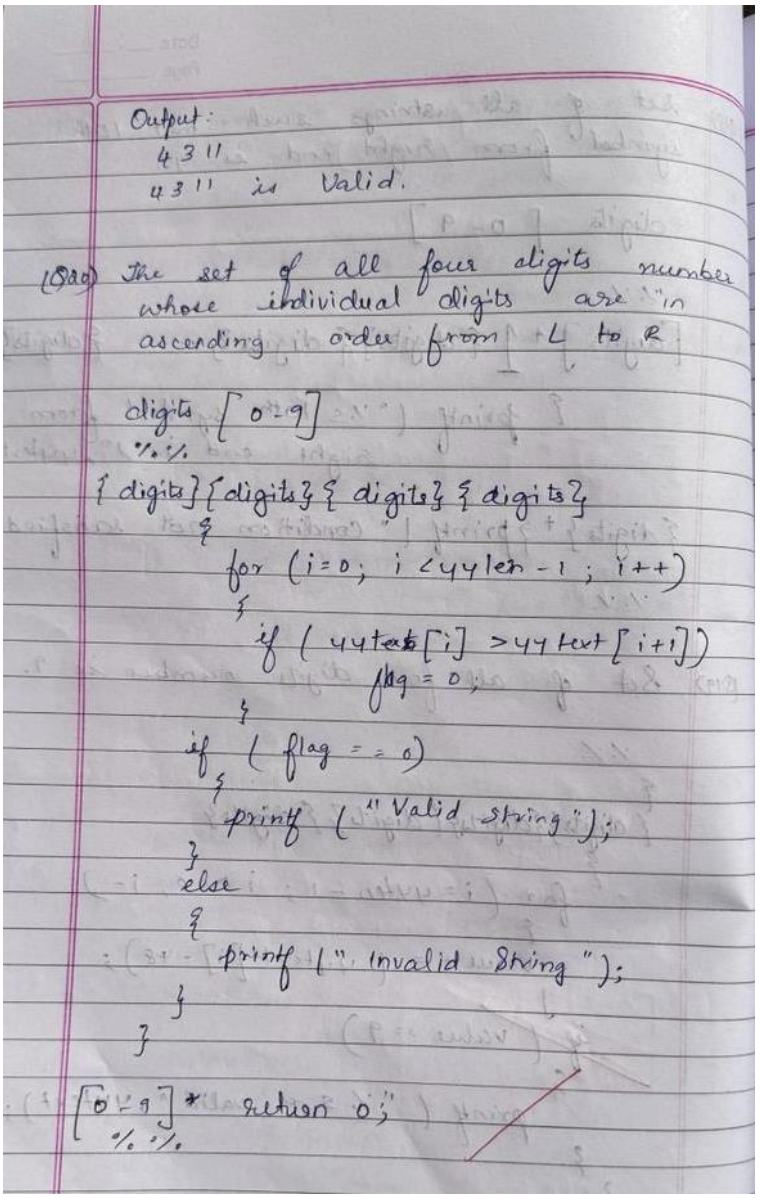
Output

```
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

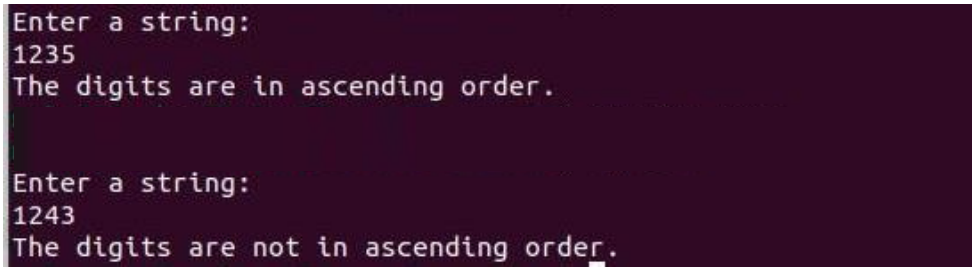
The set of all four digital numbers, whose individual digits are in ascending orderfrom left to right.

Code



```
Output:
4311
4311 is Valid.

(Qno) The set of all four digits number
whose individual digits are in
ascending order from L to R

digits [0-9]
%.%
{digits}{digits}{digits}{digits}
    {
        for (i=0; i<yylen-1; i++)
        {
            if (yytext[i] > yytext[i+1])
                flag = 0;
        }
    if (flag == 0)
    {
        printf ("Valid string");
    }
    else
    {
        printf ("Invalid String");
    }
}

[0-9]* return 0;
%.%
```

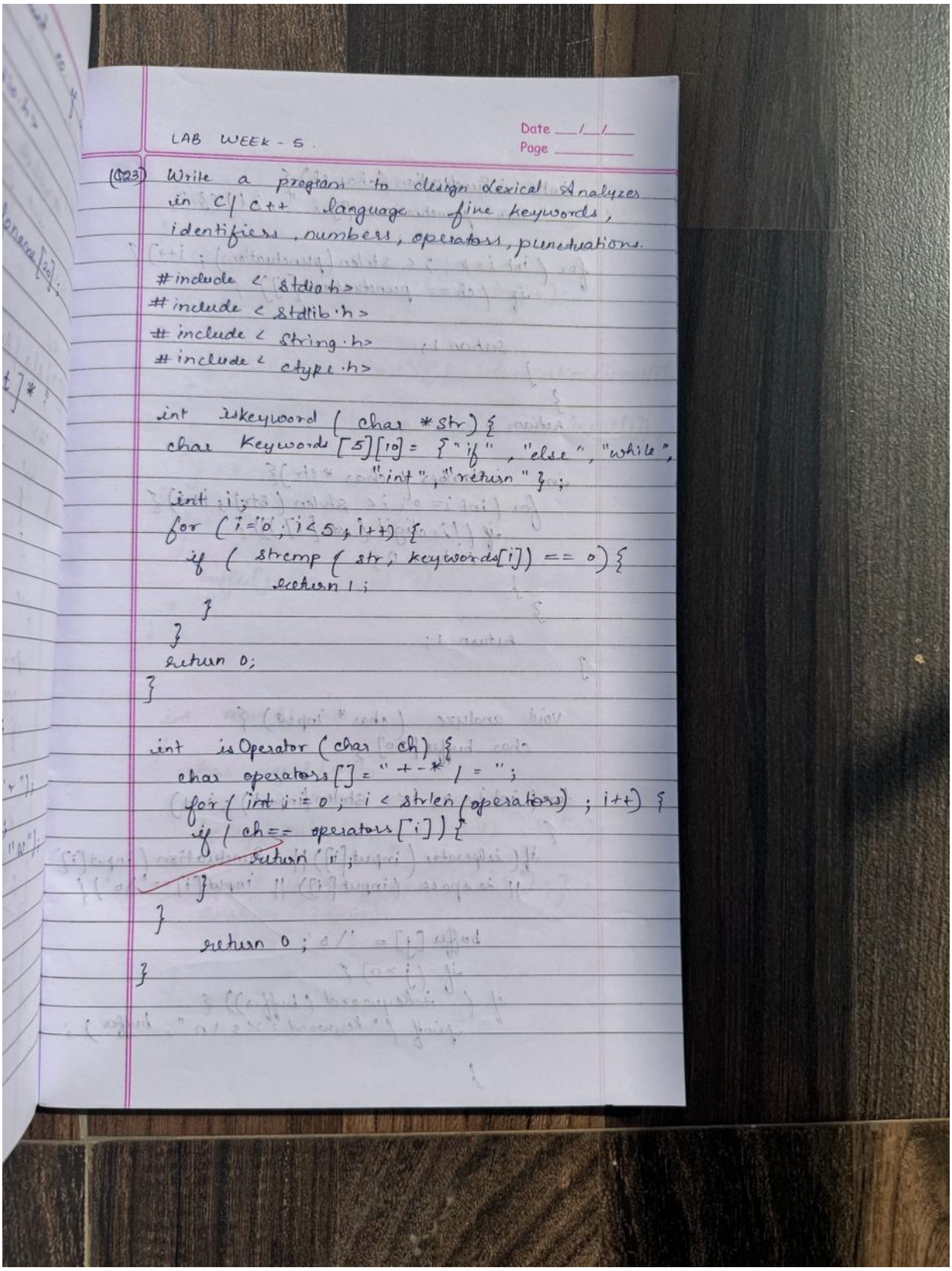
Output


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

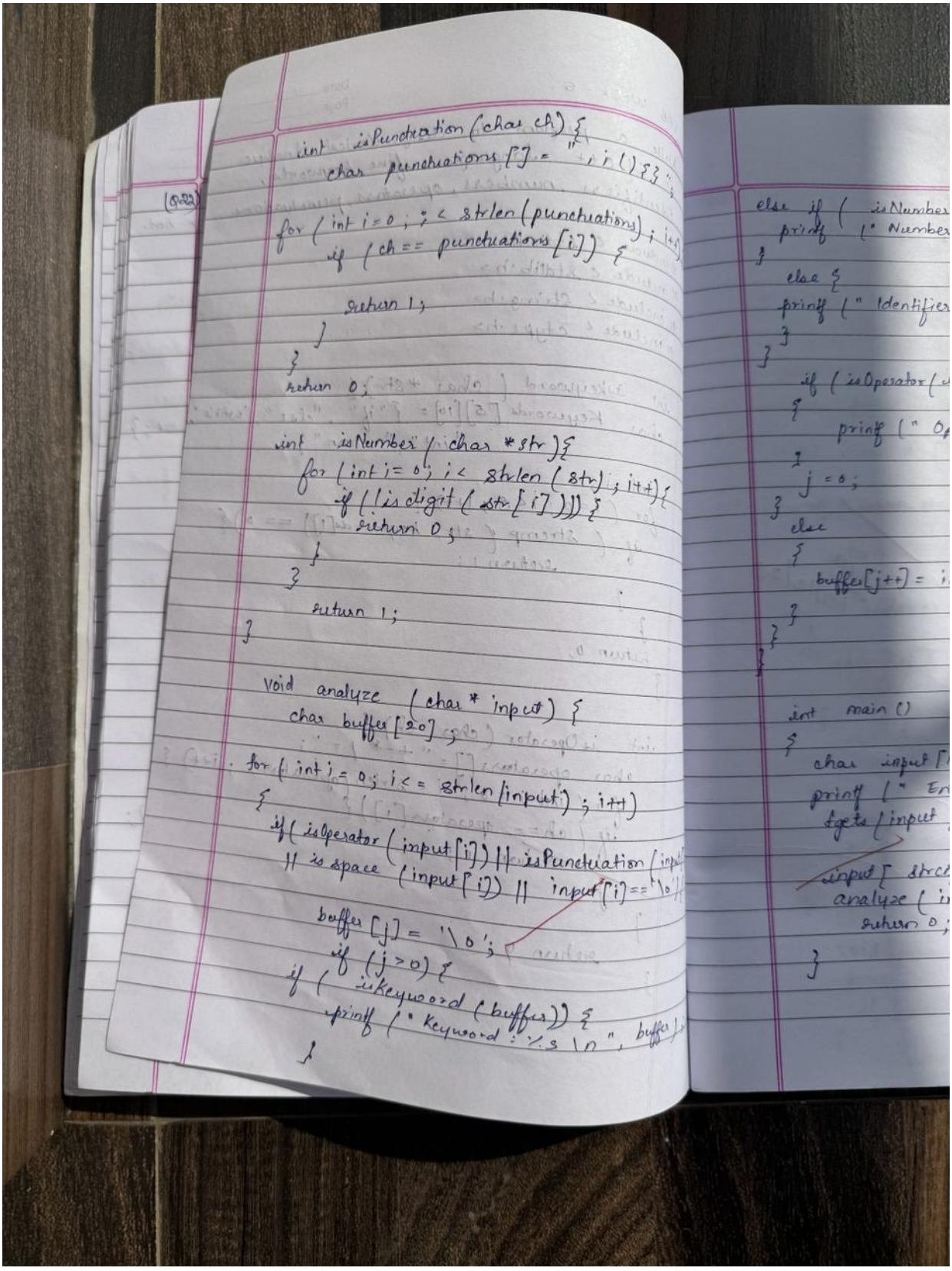


LAB WEEK - 5

Date __/__/____
Page _____

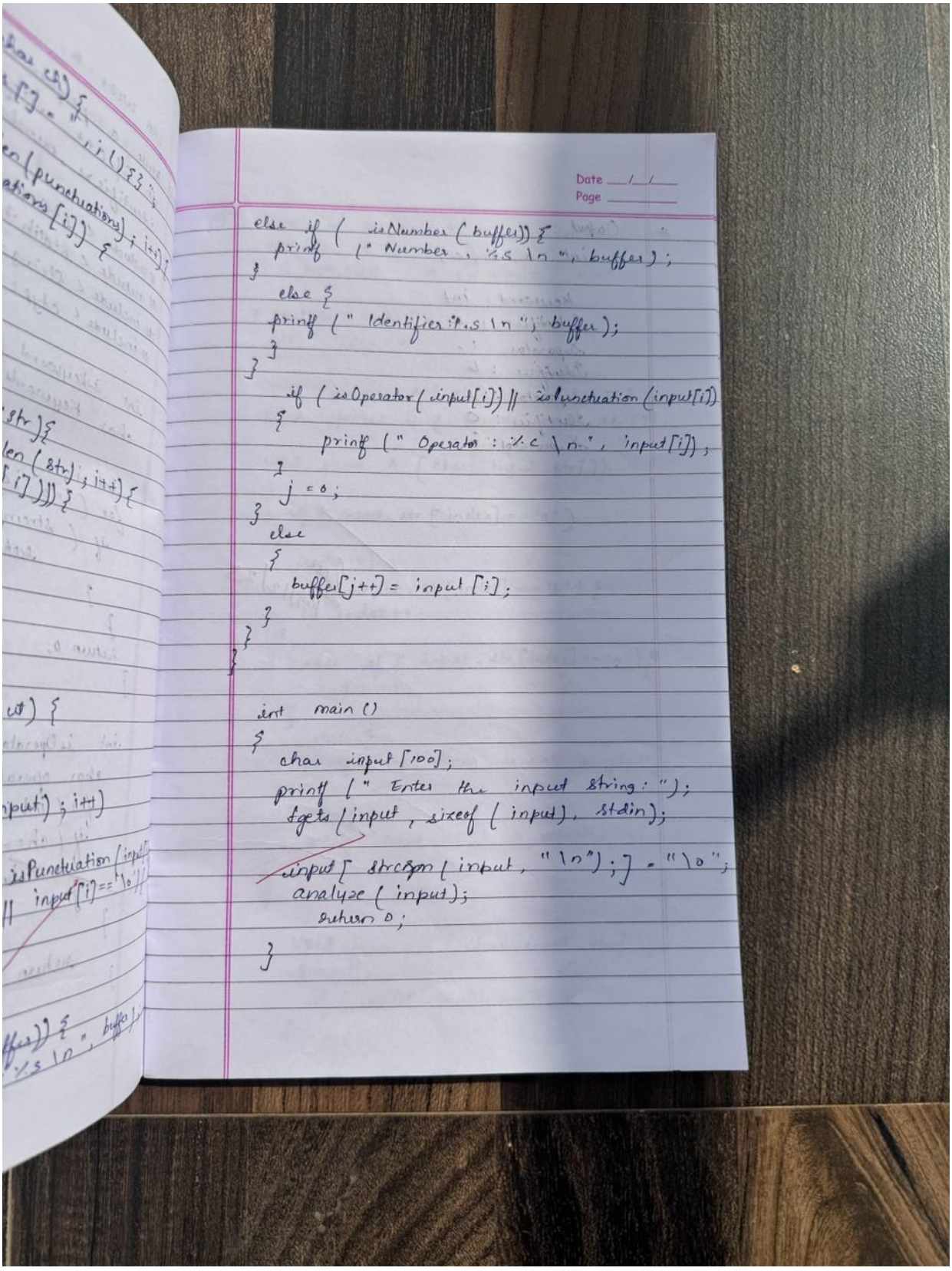
(Q23) Write a program to design lexical Analyzer in C/C++ language fine keywords, identifiers, numbers, operators, punctuations.

```
#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 (i=0; i<5; i++) {
  if (strcmp (str, keywords[i]) == 0) {
      return 1;
  }
}
return 0;
}

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



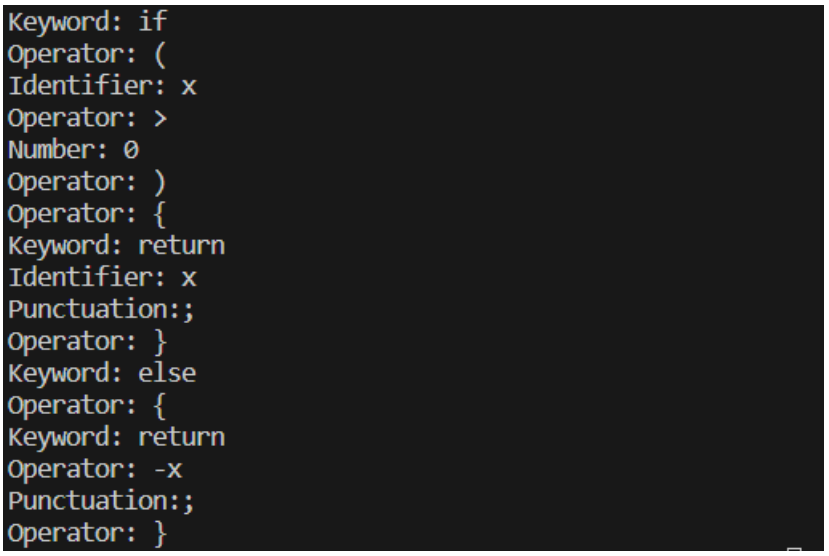
```
int isPunctuation(char ch) {
    char punctuations[] = ",;(){}";
    for (int i=0; i < strlen(punctuations); i++)
        if (ch == punctuations[i]) {
            return 1;
        }
    }
    return 0;
}

int isNumber(char *str){
    for (int i=0; i< strlen(str); i++){
        if (!isdigit(str[i])) {
            return 0;
        }
    }
    return 1;
}

void analyze(char* input) {
    char buffer[20];
    for (int i=0; i<=strlen(input); i++)
    {
        if( isOperator(input[i]) || isPunctuation(input[i])
        || isspace(input[i]) || input[i]=='\0')
        {
            buffer[j] = '\0';
            if (j>0) {
                if ( isKeyword(buffer)) {
                    printf("Keyword: %s\n", buffer);
                }
```

```
else if ( isNumber
    printf
}
else {
    printf(" Identifier
}
}
    if (isOperator(
    {
        printf(" O
    }
    j = 0;
}
else
{
    buffer[j++] = i
}
}
}

int main()
{
    char input[
    printf(" En
    fgets(input
    input[ strc
    analyze(i
    return 0;
}
```



```
else if ( isNumber (buffer)) {
  printf (" Number : %s \n ", buffer);
}
  else {
  printf (" Identifier : %s \n ", buffer);
  }
}
  if ( isOperator ( input[i]) || isPunctuation (input[i])
  {
      printf (" Operator : %c \n ", input[i]);
  }
  j = 0;
}
  else
  {
    buffer[j++] = input [i];
  }
}
}

int main ()
{
  char input [100];
  printf (" Enter the input string : ");
  fgets (input, sizeof ( input), stdin);

  input [ strcspn ( input, "\n") ; ] = "\0";
  analyze ( input);
    return 0;

}
```

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: }

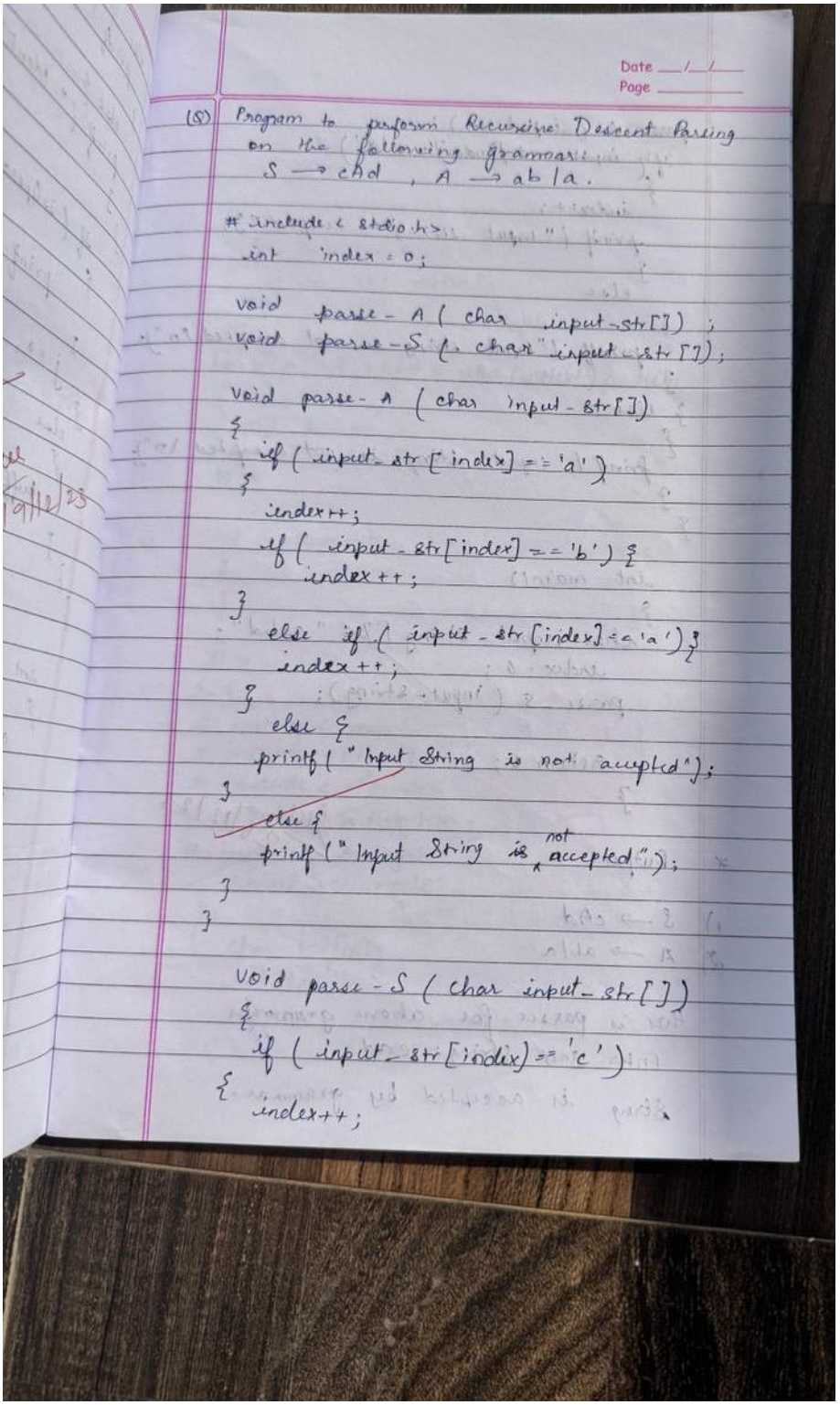
Lab 6

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

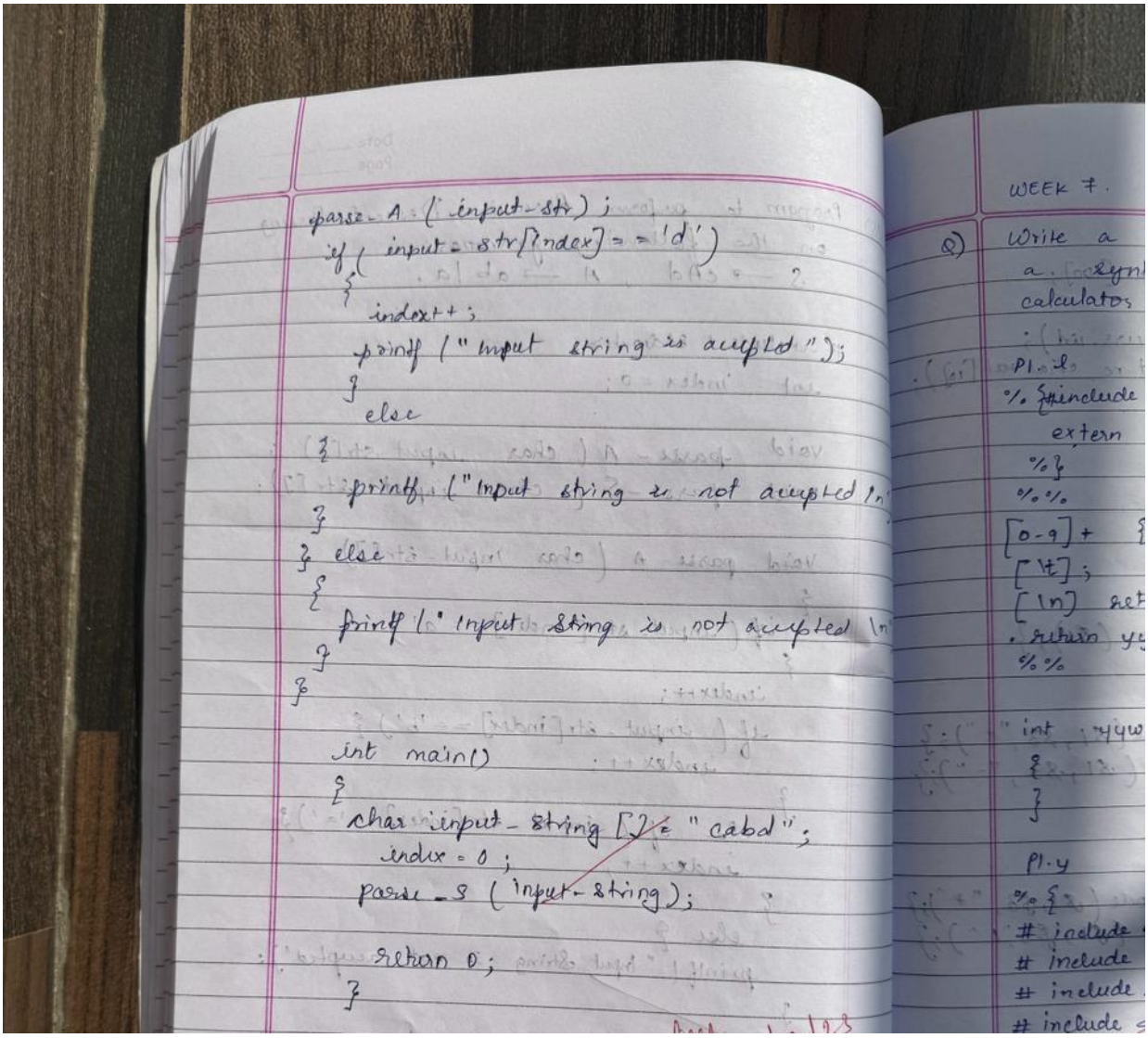
S->cAd

A->ab | a

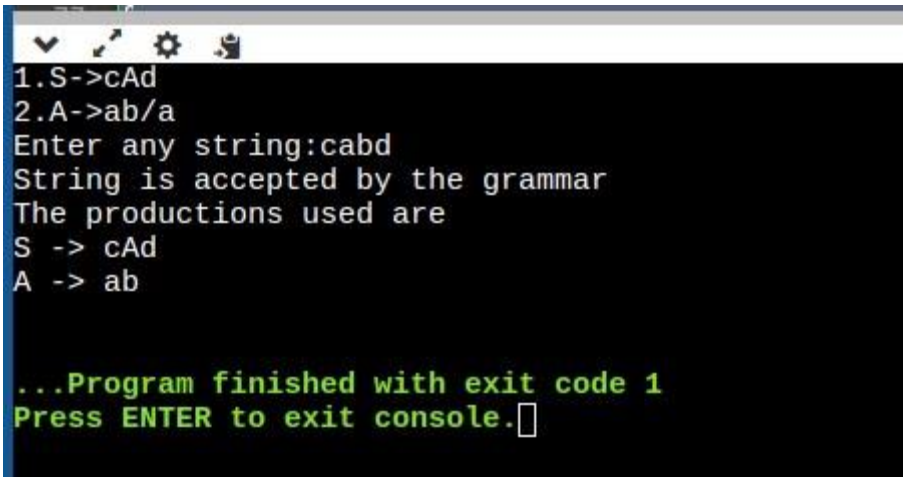
Code



```
(6) Program to perform Recursive Descent Parsing
on the following grammar:
S → cAd , A → ab | a.

#include < stdio.h>
int index = 0;

void parse-A ( char input-str[] ) ;
void parse-S ( char input-str[] );

void parse-A ( char input-str[])
{
  if ( input-str[index] == 'a' )
  {
    index++;
    if ( input-str[index] == 'b' ) {
      index++;
  }
    else if ( input-str[index] == 'a' ) {
    index++;
  }   else {
    printf ( " Input String is not accepted" );
  }
    else {
    printf ( " Input String is not accepted." );
  }
}

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
else
printf ("input string is not accepted
int main()
char input_string [] = "cabd";
indix = 0;
parse_S (input_string);
return 0;
WEEK 7.
Q) Write a
calculator
%{ #include
extern
%}
%%
[0-9]+
[\t];
[\n]
%%
int
%{
# include
# include
# include

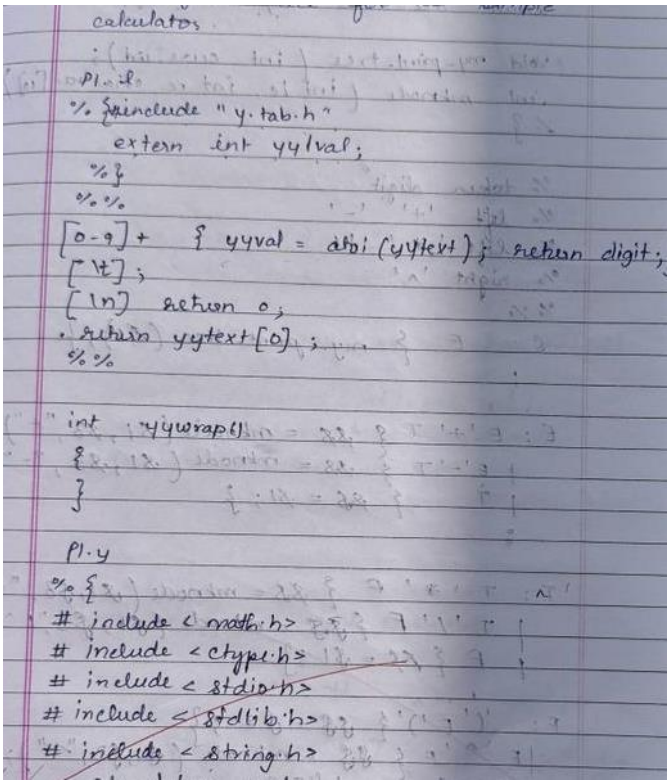
Output


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.

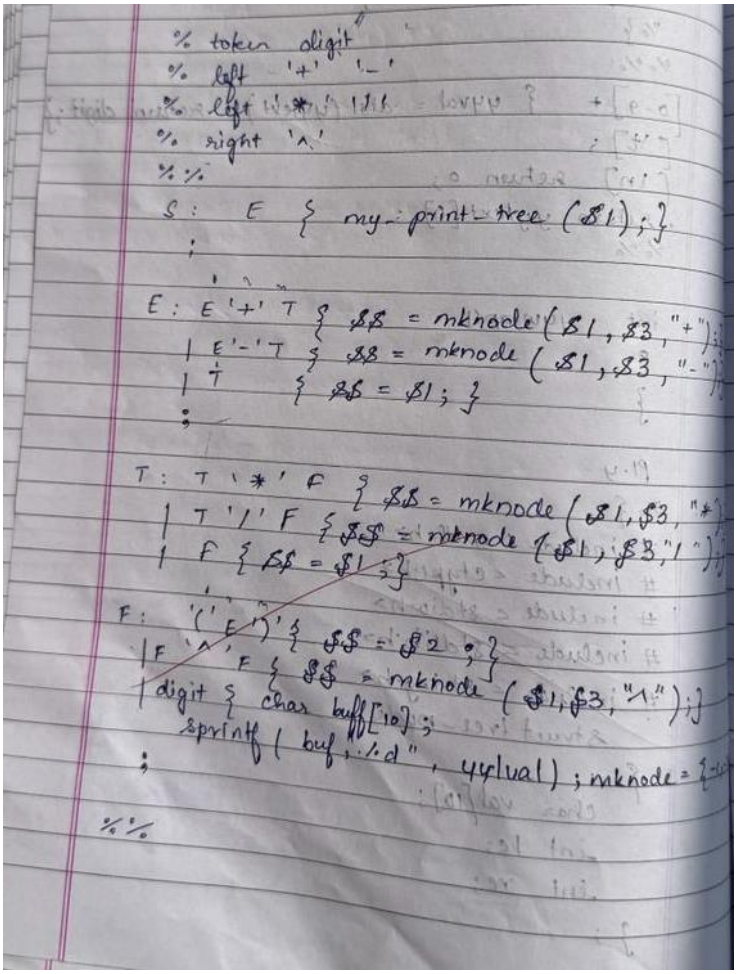
Lab 7

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

Code



```
calculator

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>
```

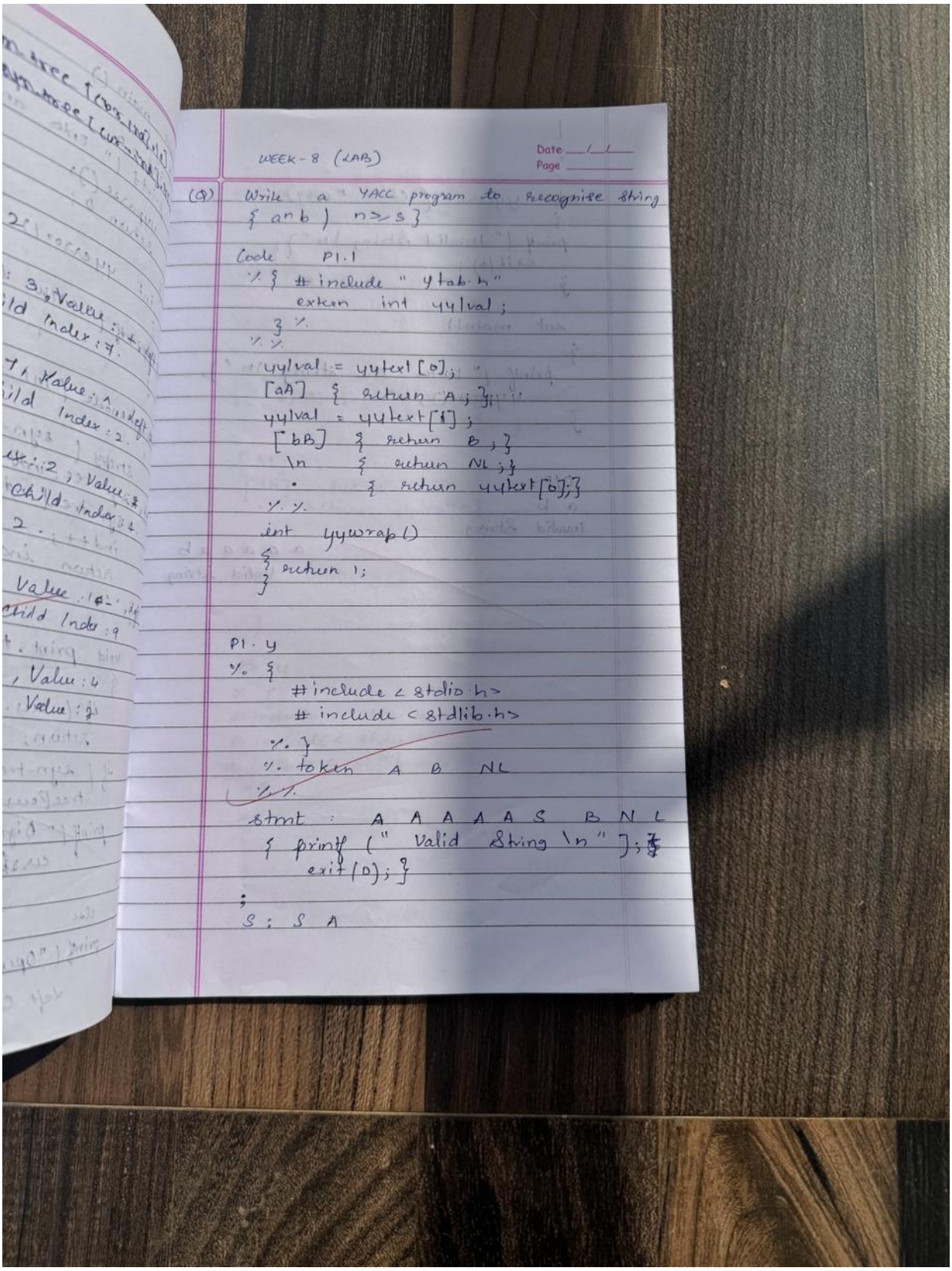


Output

```
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



WEEK-8 (LAB)

Date __/__/__
Page _____

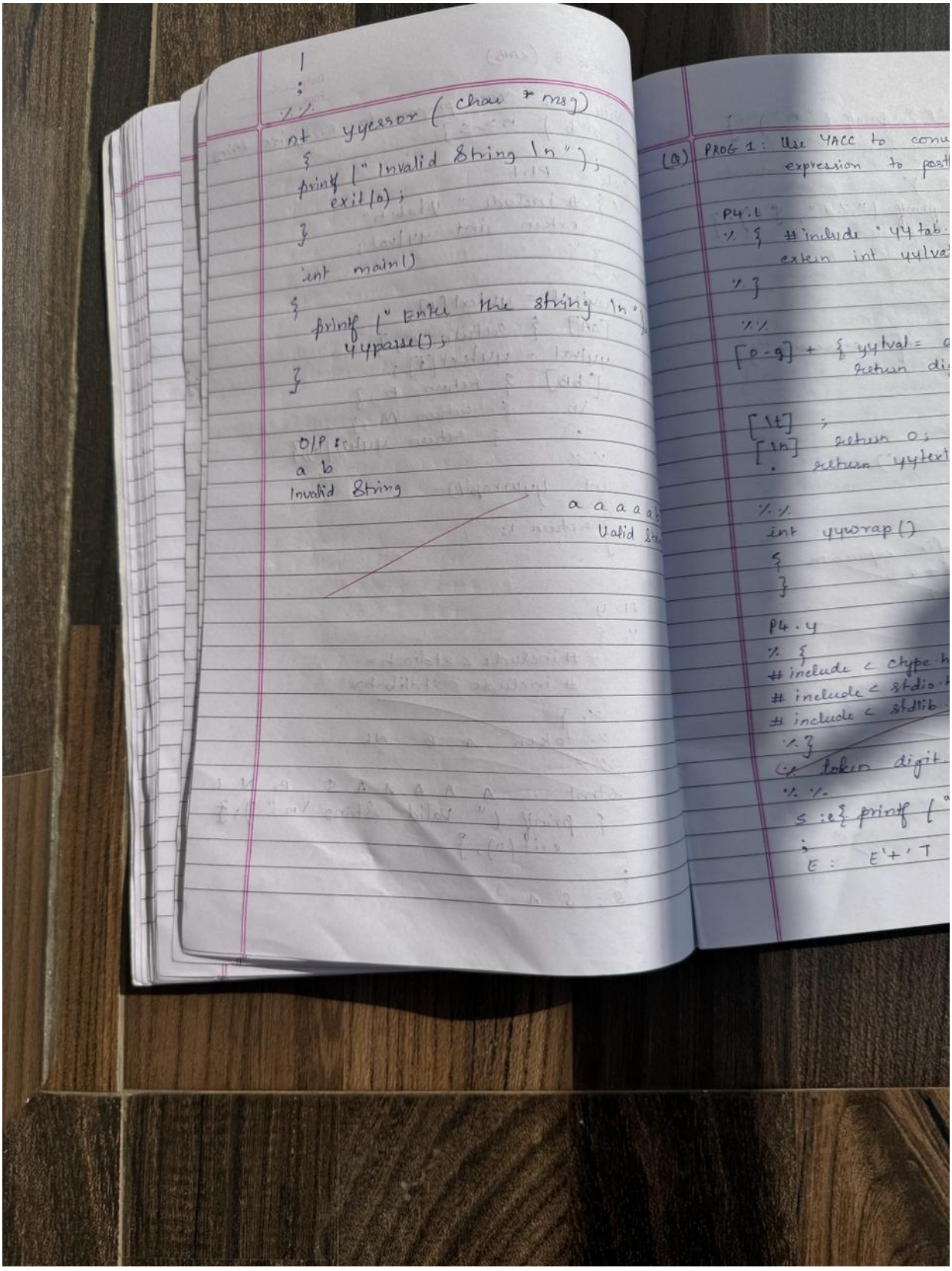
(Q) Write a YACC program to recognise string { a^n b | n>=5 }

Code P1.l

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

P1.y

```
%{
   #include <stdio.h>
   #include <stdlib.h>
%}
%token A B NL
%%
stmt : A A A A A S B N L
{ printf("Valid String \n"); exit(0); }
;
S : S A
```

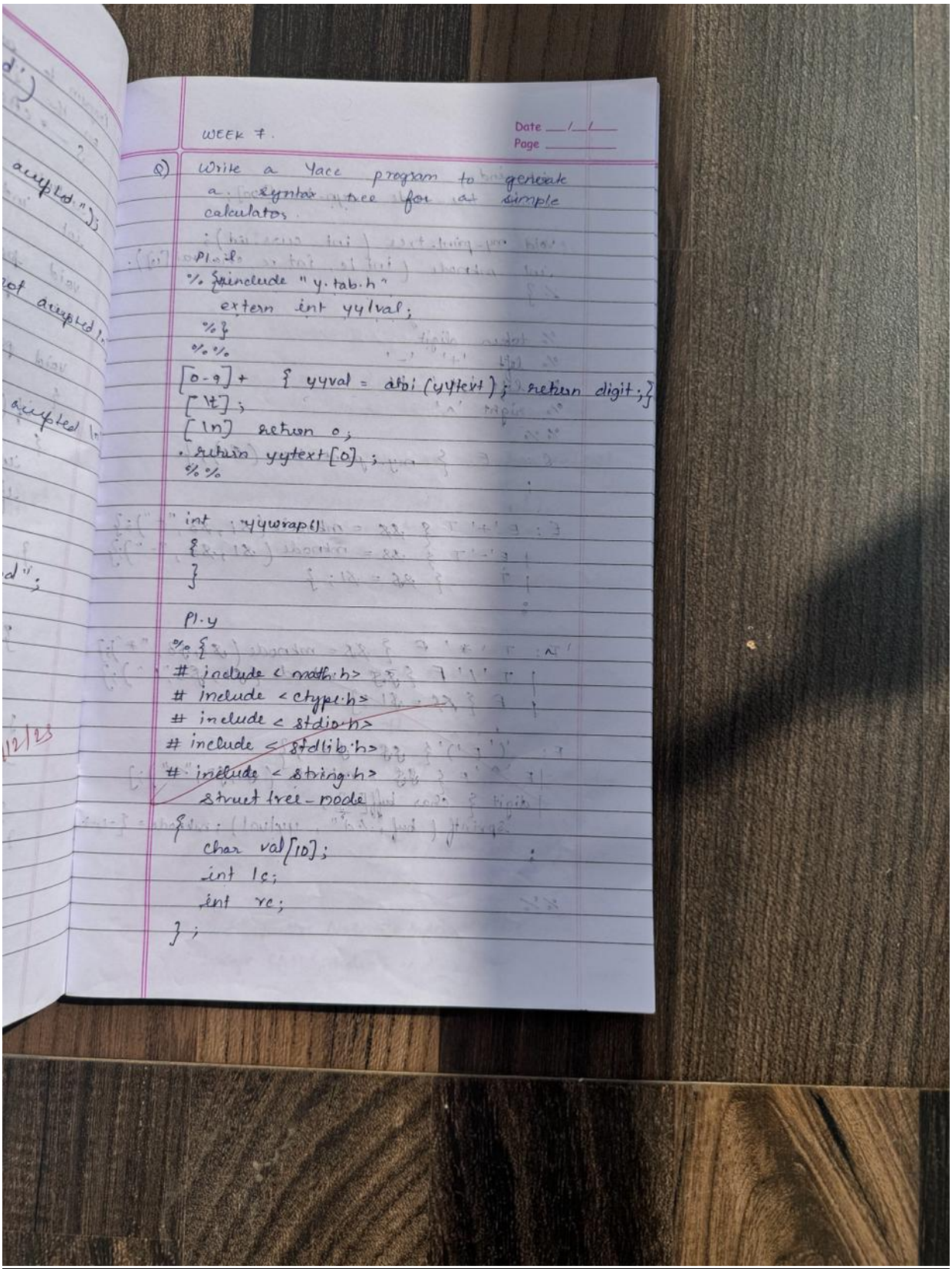

yyerror (char *msg)
printf ("Invalid String \n");
exit(0);
int main()
printf ("Enter the string \n");
yyparse();
O/P :
a b
Invalid String
PROG 1: Use YACC to
expression to
P4.1
#include "yy.tab.
extern int yylval
[0-9] + { yylval =
return di
[\t] ;
[\n] return 0;
return yytext
int yywrap()
P4.4
#include <ctype.h
#include <stdio.
#include <stdlib
token digit
E : E'+' T

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



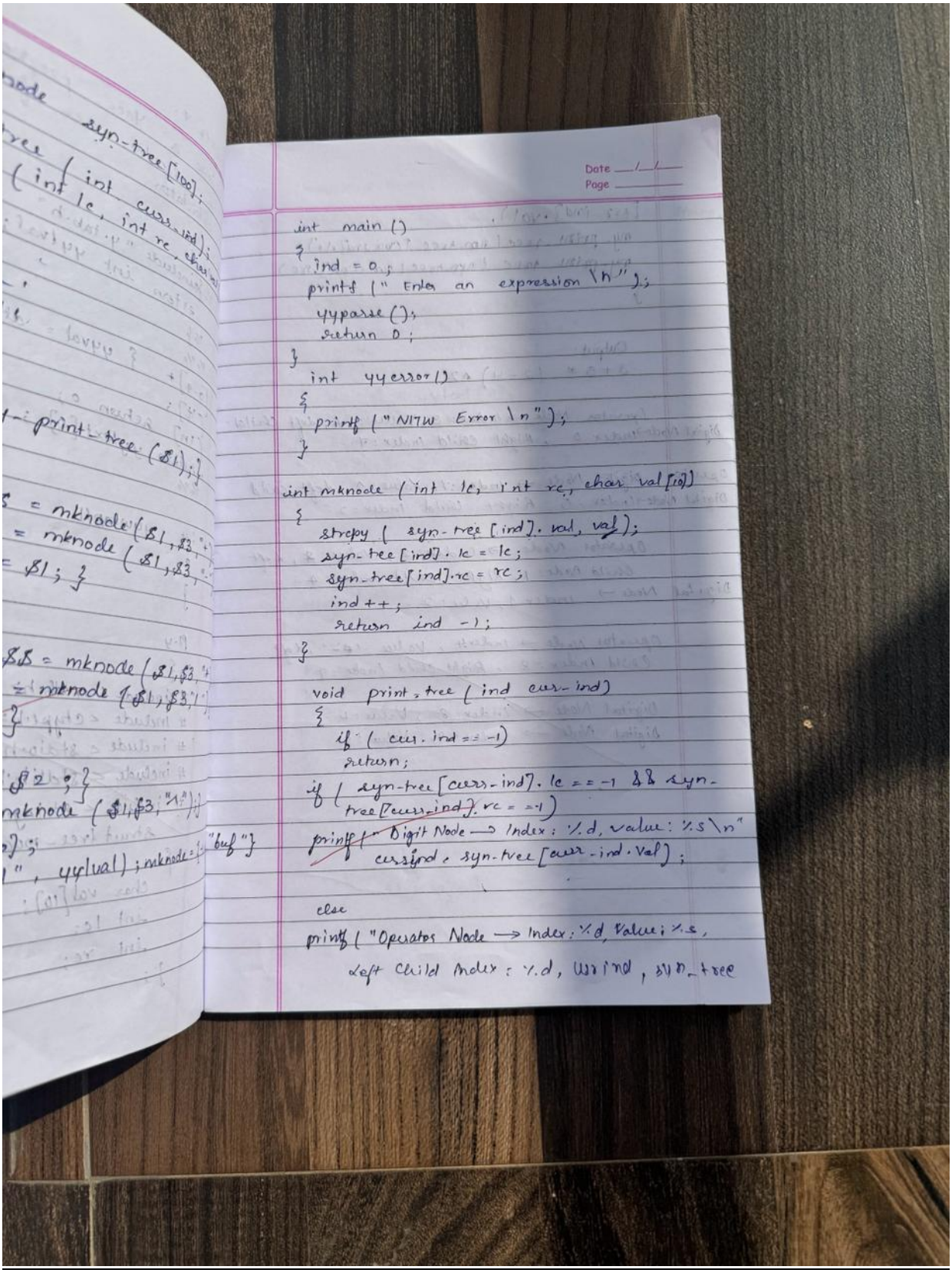
WEEK 7.

Date __/__/__
Page _____

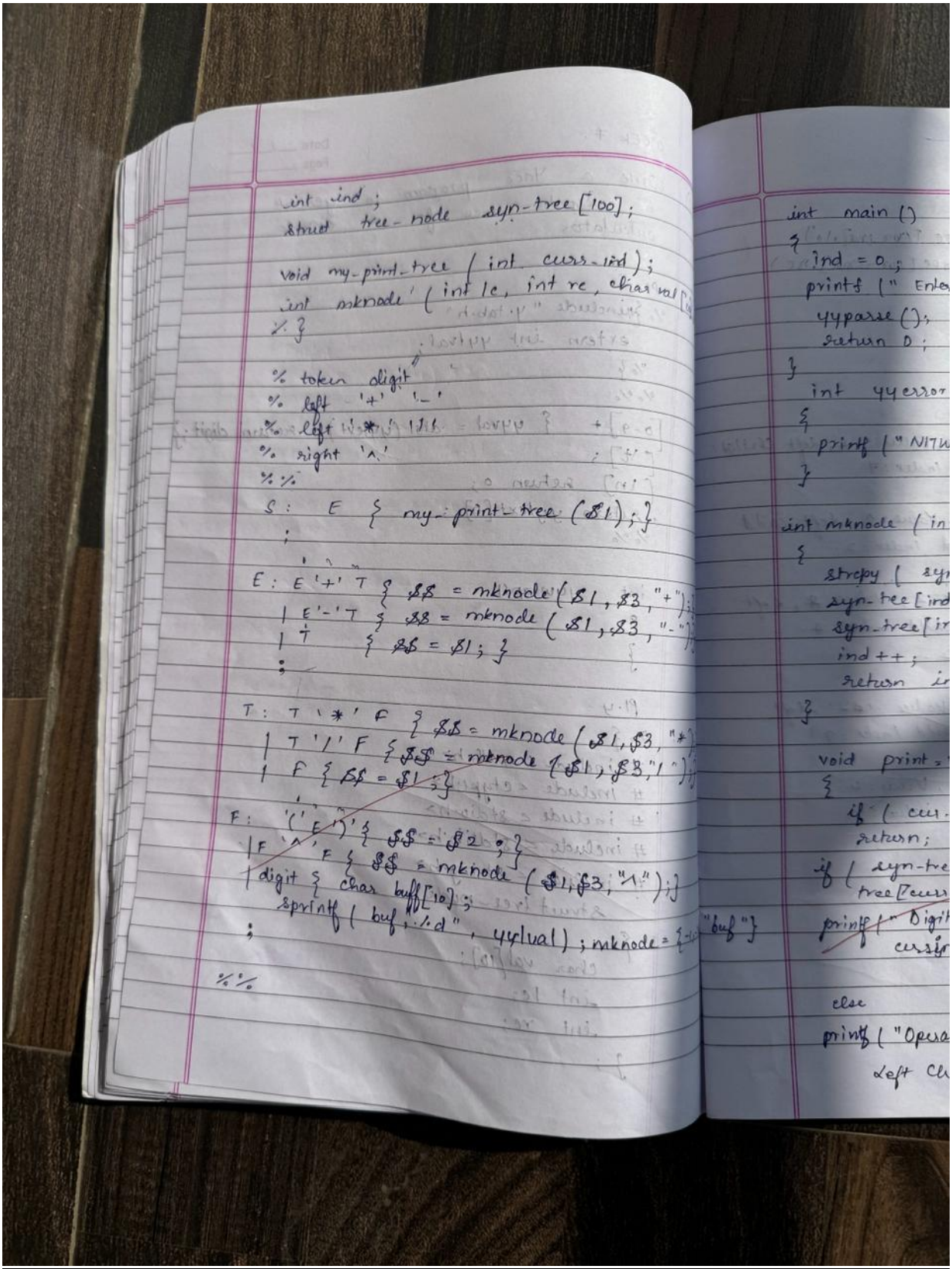
Q) Write a Yacc program to generate a syntax tree for a simple calculator.

```
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;
};
```

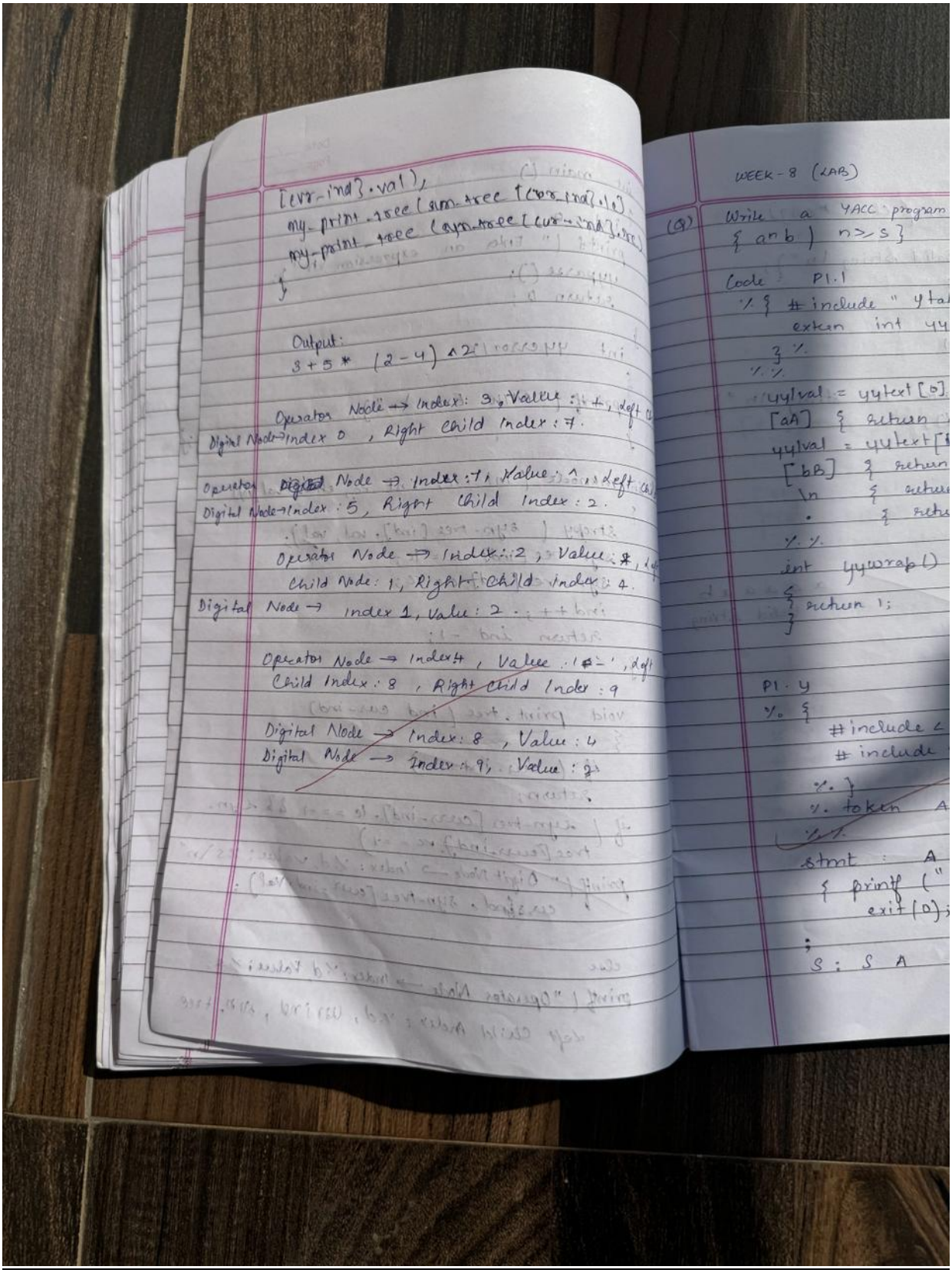


```
int main ()
{ ind = 0;
  printf (" Enter an expression \n");
   yyparse ();
   return 0;
}
  int yyerror ()
  {
   printf (" NITW Error \n");
  }

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 print_tree ( ind curr_ind)
   {
     if ( curr. ind == -1)
     return;
   if ( syn_tree [curr_ind]. lc == -1 && syn_
      tree [curr_ind]. rc == -1)
   printf (" Digit Node -> Index : %d, value : %s \n"
        curr_ind, syn_tree [curr_ind. val);

   else
   printf ( "Operator Node -> Index : %d, Value : %s,
      Left Child Index : %d, %s ind, syn_tree
```



```
int ind;
struct tree_node syn_tree[100];

void my_print_tree(int curr_ind);
int mknode(int lc, int rc, char val[
%}

% 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(buf, "%d", yylval); mknode = {
  "buf" }
;

%%
```

```
int main()
{ ind = 0;
printf(" Ente
yyparse();
return 0;
}
int yyerror
{
printf("NIT
}

int mknode(in
{
strcpy( syn
syn_tree[ind
syn_tree[in
ind++;
return i
}

void print
{
if( cur
return;
if( syn_tre
tree[curr
printf(" Digit
cursi

else
printf("Opera
Left Cu
```


[cur_ind].val);
my_print_tree(syn_tree[cur_ind].l
my_print_tree(syn_tree[cur_ind].r
}
Output:
3 + 5 * (2 - 4) ^ 2
Operator Node → Index: 3, Value: +, left
Digital Node→Index 0, Right child Index: 7.
Operator Node → Index: 7, Value: ^, left
Digital Node→Index: 5, Right Child Index: 2.
Operator Node → Index: 2, Value: *,
Child Node: 1, Right Child Index: 4.
Digital Node → Index 1, Value: 2
Operator Node → Index 4, Value: '-',
Child Index: 8, Right Child Index: 9
Digital Node → Index: 8, Value: 4
Digital Node → Index: 9, Value: 2
WEEK-8 (LAB)
(Q) Write a YACC program
{ a^n b | n>=5 }
Code P1.1
%{ #include " y.tab
extern int yy
%}
%%
yylval = yytext[0]
[aA] { return
yylval = yytext[
[bB] { return
\n { return
. { return
%%
int yywrap()
{ return 1;
}
P1.y
%{
#include <
#include
%}
%token A
%%
stmt : A
{ printf("
exit(0);
;
S : S A

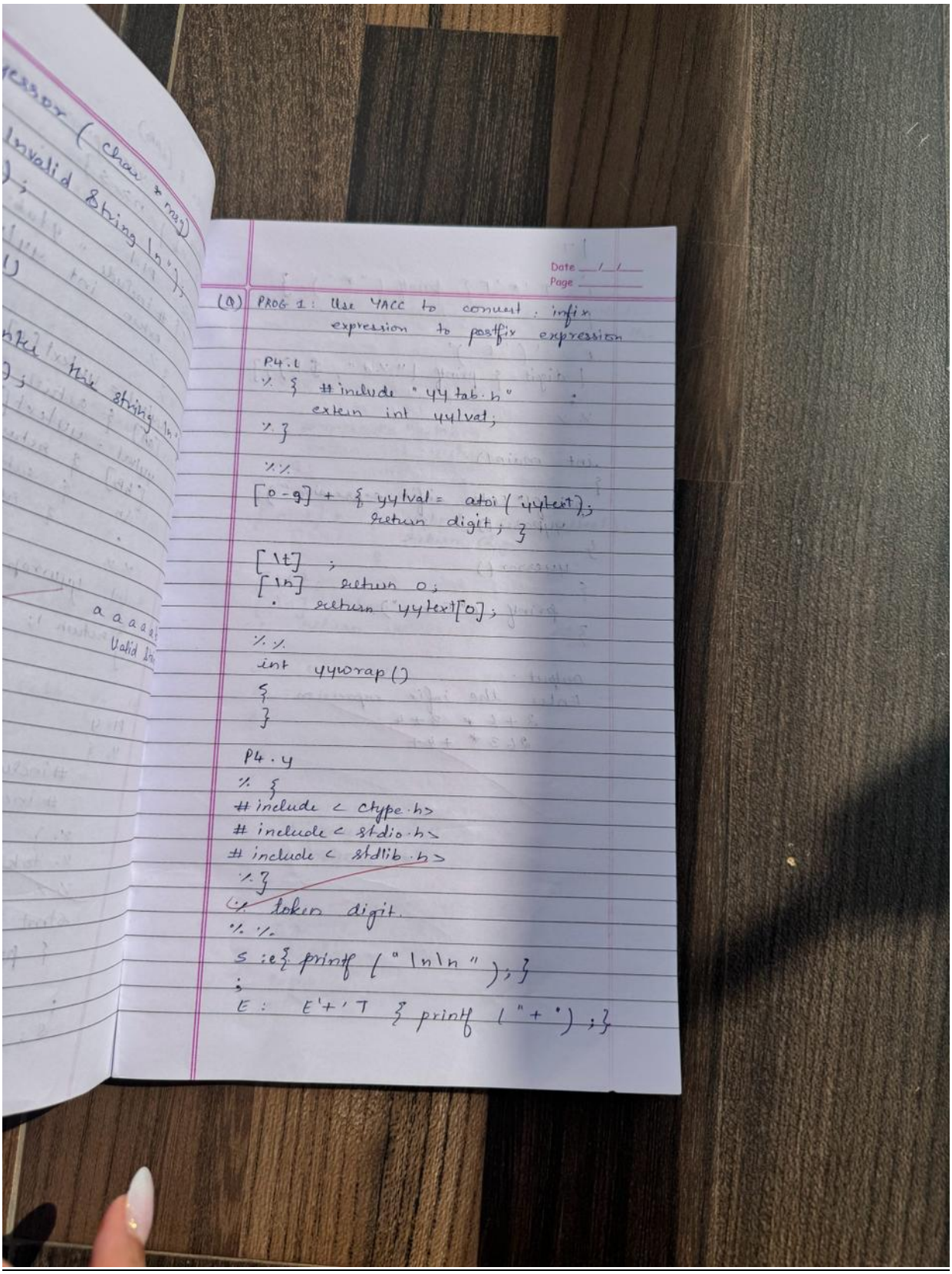
Output

```
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

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

Code



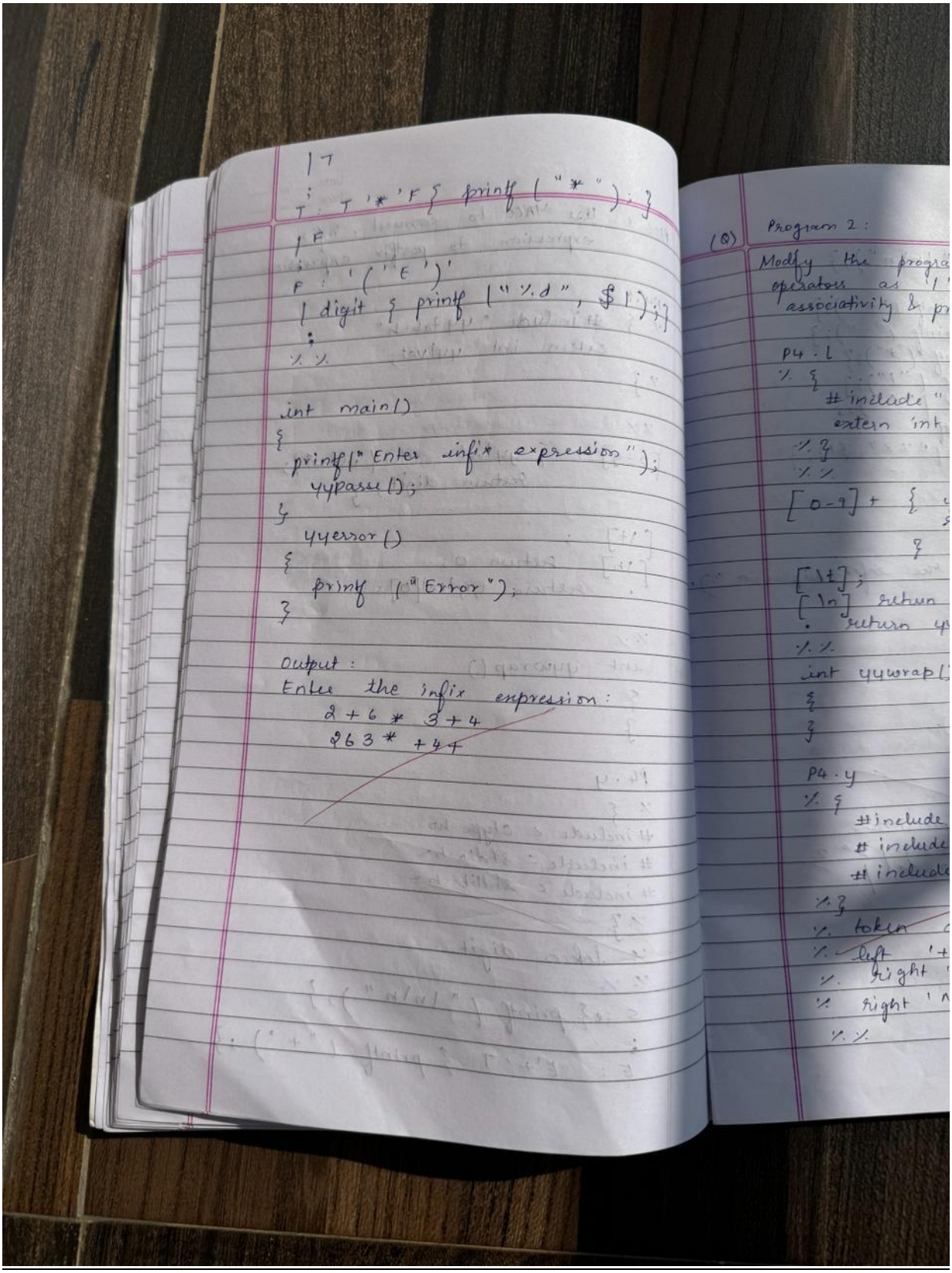
(a) PROG 1: Use YACC to convert infix expression to postfix expression

P4.l

```
%{ #include "y.tab.h"
   extern int yylval;
%}

%%
[0-9]+ { yylval = 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"); }
;
E : E'+'T { printf("+"); }
```

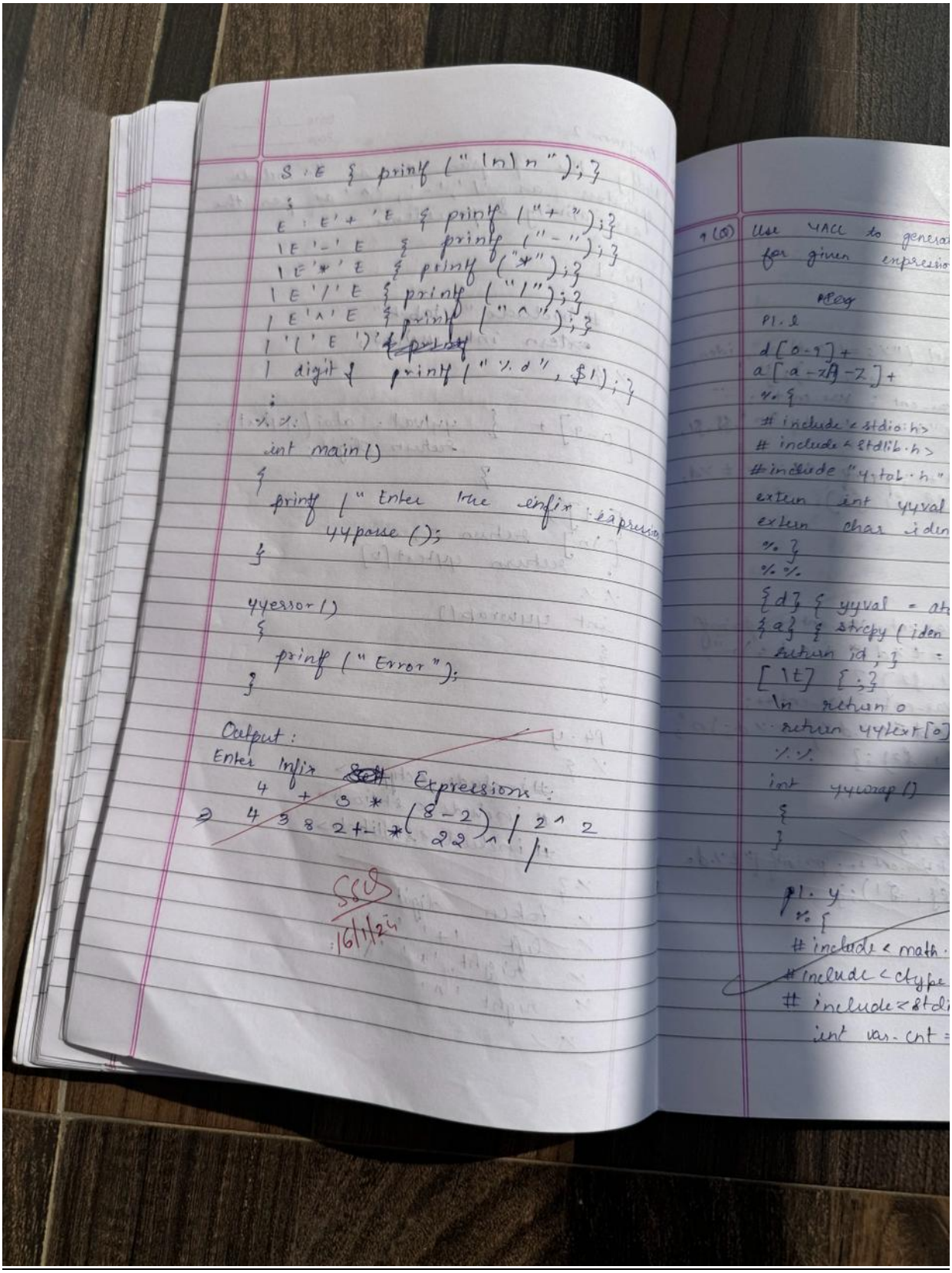


```
| T
;
T : T '*' F { printf ("*"); }
| F
;
F : '(' E ')'
| digit { printf ("%d", $1); }
;
%%

int main()
{
  printf("Enter infix expression");
  yyparse();
}
  yyerror()
{
  printf ("Error");
}

Output:
Enter the infix expression:
  2 + 6 * 3 + 4
  2 6 3 * + 4 +
```

(8) Program 2:

Modify the progra
operators as '
associativity & pr

```
p4.l
%{
  #include "
  extern int
%}
%%
[0-9]+ {
}
[\t];
[\n] return
. return y
%%
int yywrap()
{
}

p4.y
%{
  #include
  #include
  #include
%}
%token
%left '+
%right
%right '^
%%
```



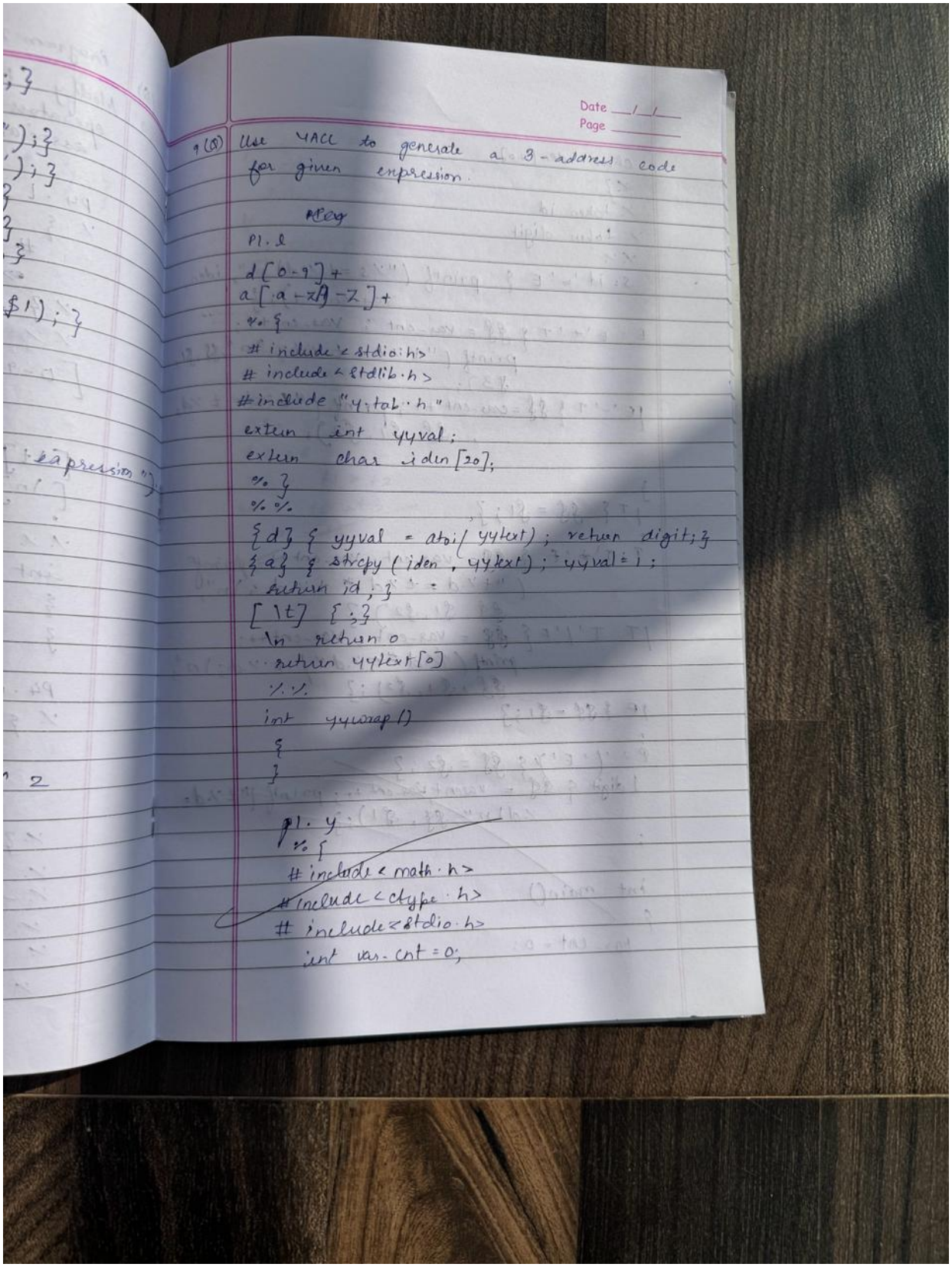
Output

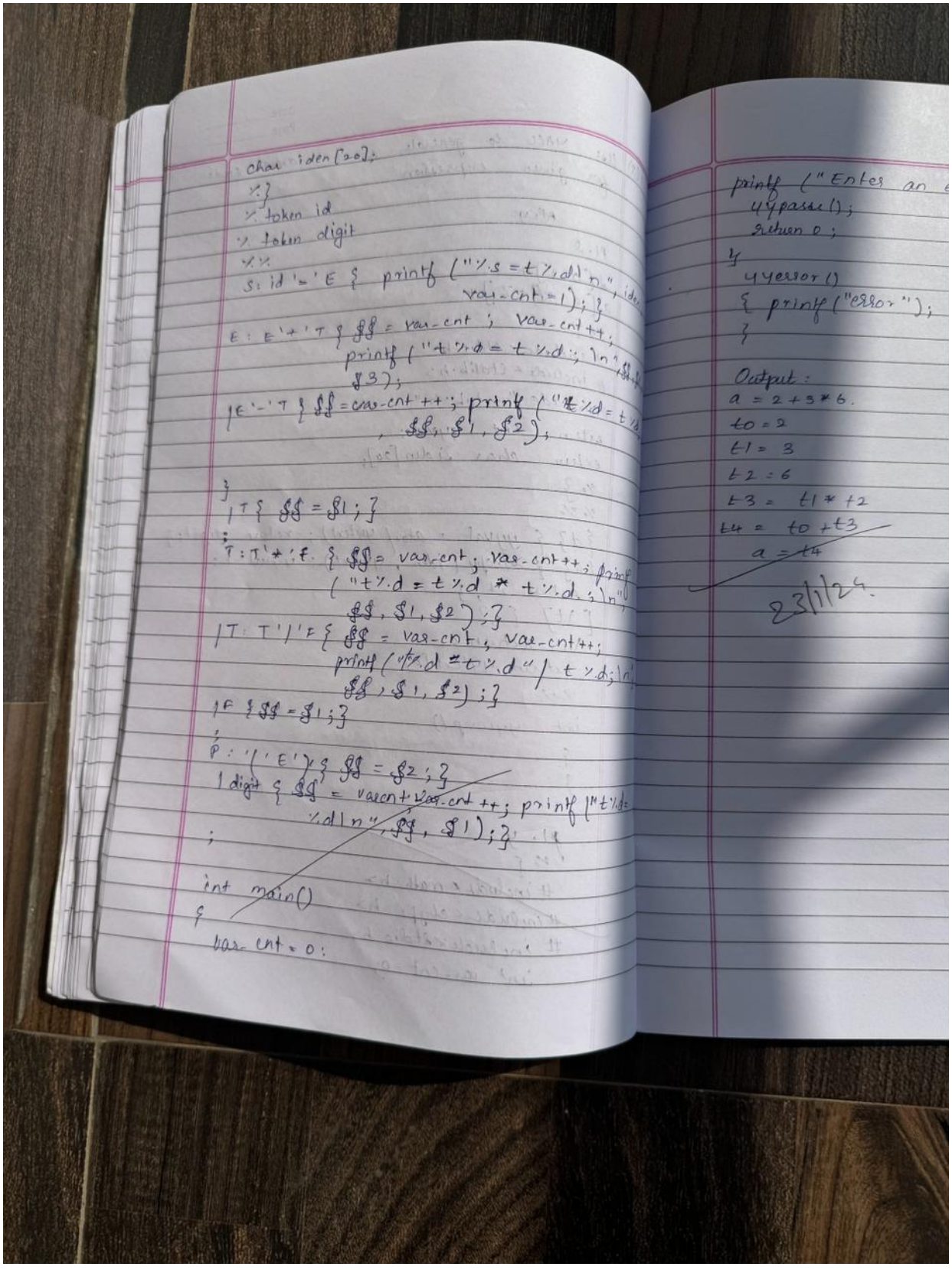
```
Enter an infix expression:
2+3*8/4^3-3
238*43^/+3-
```

Lab 9

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

Code





```
char iden[20];
%}
% token id
% token digit
%%
S: id '=' E { printf("%s = t%d\n", iden, var_cnt-1); }
E: E '+' T { $$ = var_cnt; var_cnt++;
             printf("t%d = t%d + t%d\n", $$, $1, $3); }
 | E '-' T { $$ = var_cnt++; printf("t%d = t%d - t%d\n", $$, $1, $2); }
 | T { $$ = $1; }
;
T: T '*' F { $$ = var_cnt; var_cnt++; printf("t%d = t%d * t%d\n", $$, $1, $2); }
 | T '/' F { $$ = var_cnt; var_cnt++; printf("t%d = t%d / t%d\n", $$, $1, $2); }
 | F { $$ = $1; }
;
F: '(' E ')' { $$ = $2; }
 | digit { $$ = varcnt; var_cnt++; printf("t%d = %d\n", $$, $1); }
;

int main()
{
var_cnt = 0;
```

```
printf("Enter an
yyparse();
return 0;
}
yyerror()
{ printf("error");
}
```

Output:

a = 2 + 3 * 6.

t0 = 2

t1 = 3

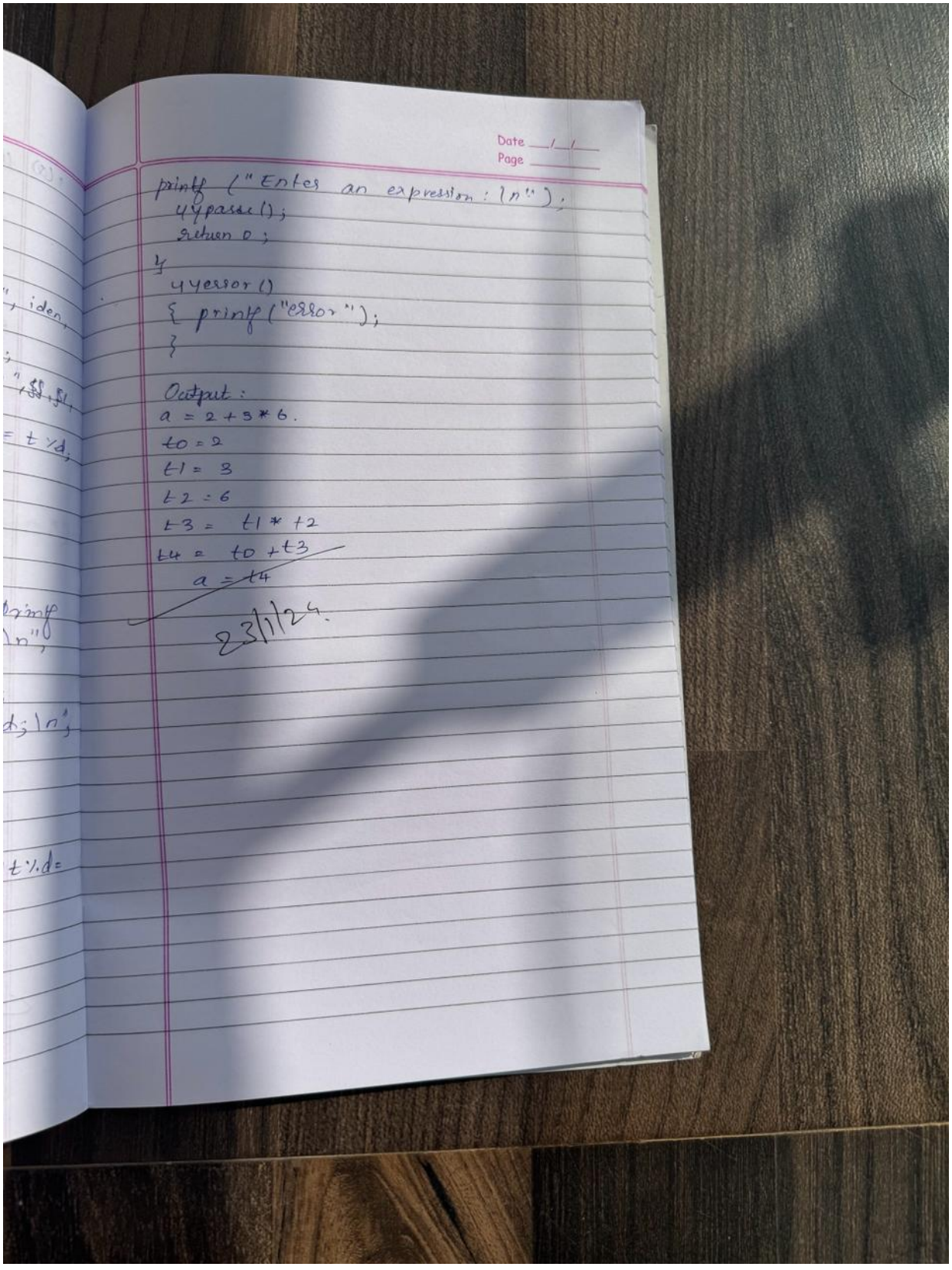
t2 = 6

t3 = t1 * t2

t4 = t0 + t3

a = t4

23/1/24.



Date __/__/__
Page _____

```
printf ("Enter an expression : \n");
yypasse();
return 0;
}
yyerror()
{ printf("error");
}
```

Output :

a = 2 + 3 * 6.

t0 = 2

t1 = 3

t2 = 6

t3 = t1 * t2

t4 = t0 + t3

a = t4

23/1/24.

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
```