VISVESVARAYA TECHNOLOGICAL UNIVERSITY

"JnanaSangama", Belgaum -590014, Karnataka.

LAB REPORT

on

COMPILER DESIGN

Submitted by

MANASA SRIRAMAIAH(1BM21CS101)

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 **Manasa Sriramaiah(1BM21CS101)**, 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

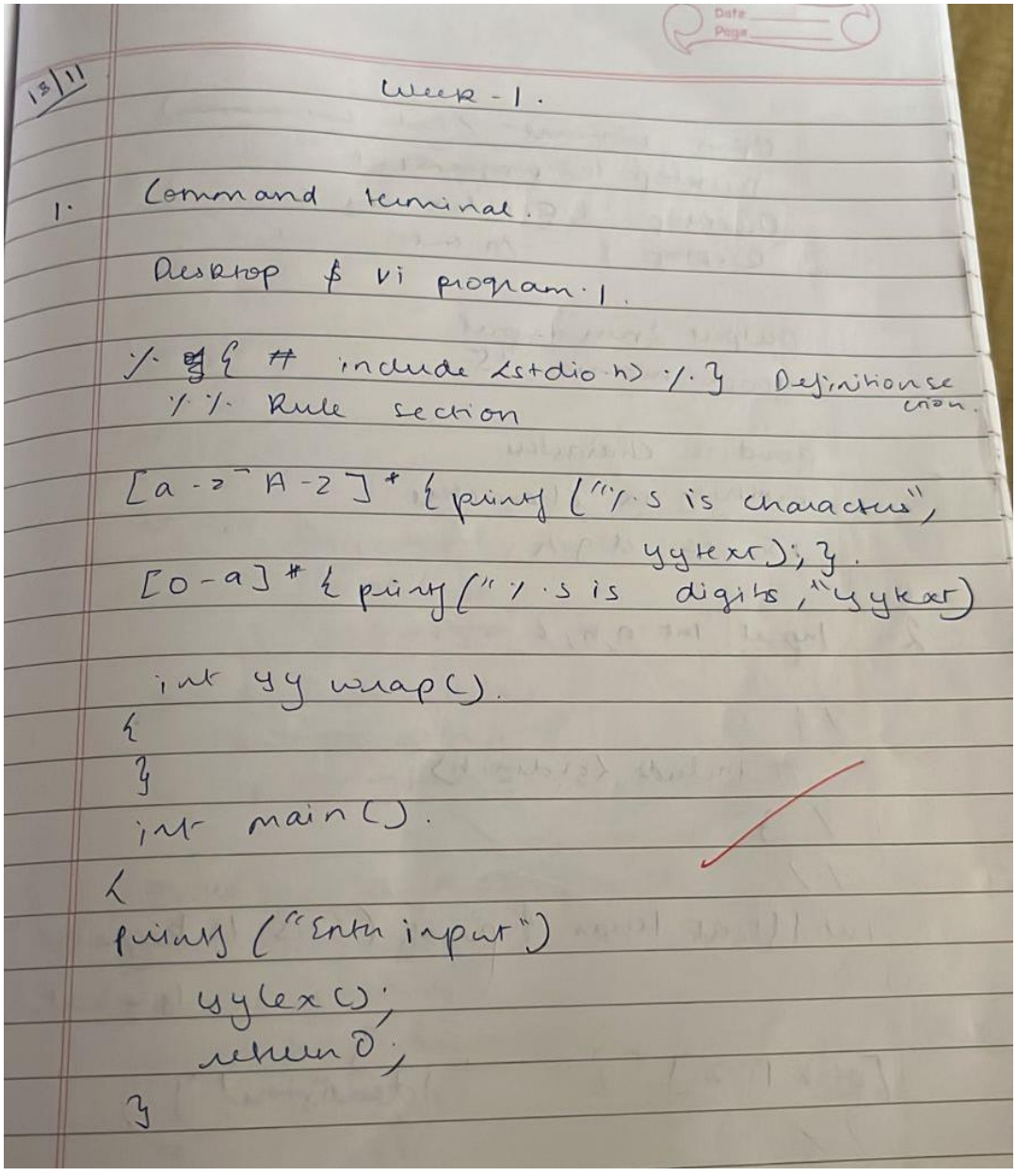
I, Manasa sriramaiah (1BM21CS101), student of 5th Semester, B.E, Department of Computer Science and Engineering, B. M. S. College of Engineering, Bangalore, here by declare that, this lab report entitled " **Compiler Design**" has been carried out by me under the guidance of Prof. Prameetha Pai,, Assistant 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 count the number of characters and digits in a string.

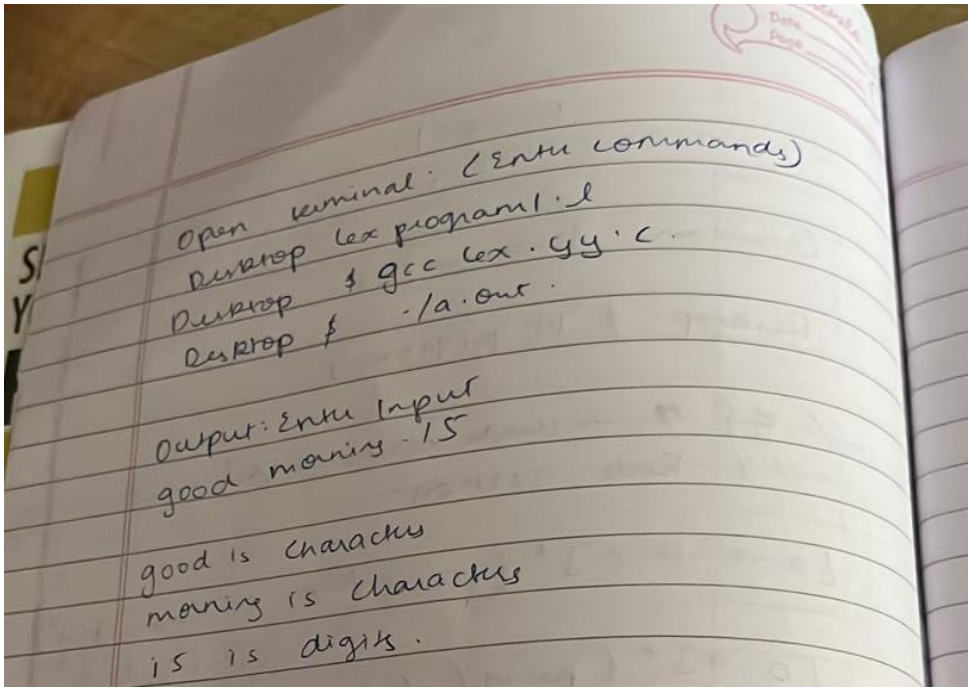


Week - 1.

1. Command terminal.

Desktop $ vi program.l

```
%{ # include <stdio.h> %} Definitionsection.
%% Rule section

[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;
}
```

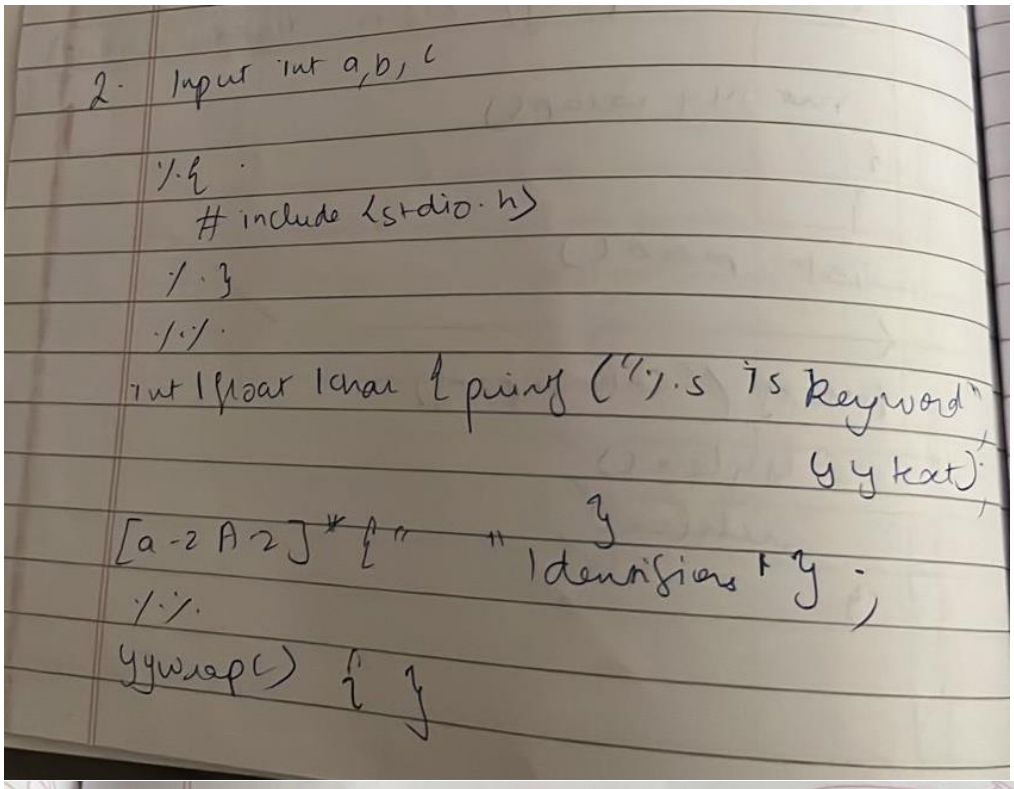


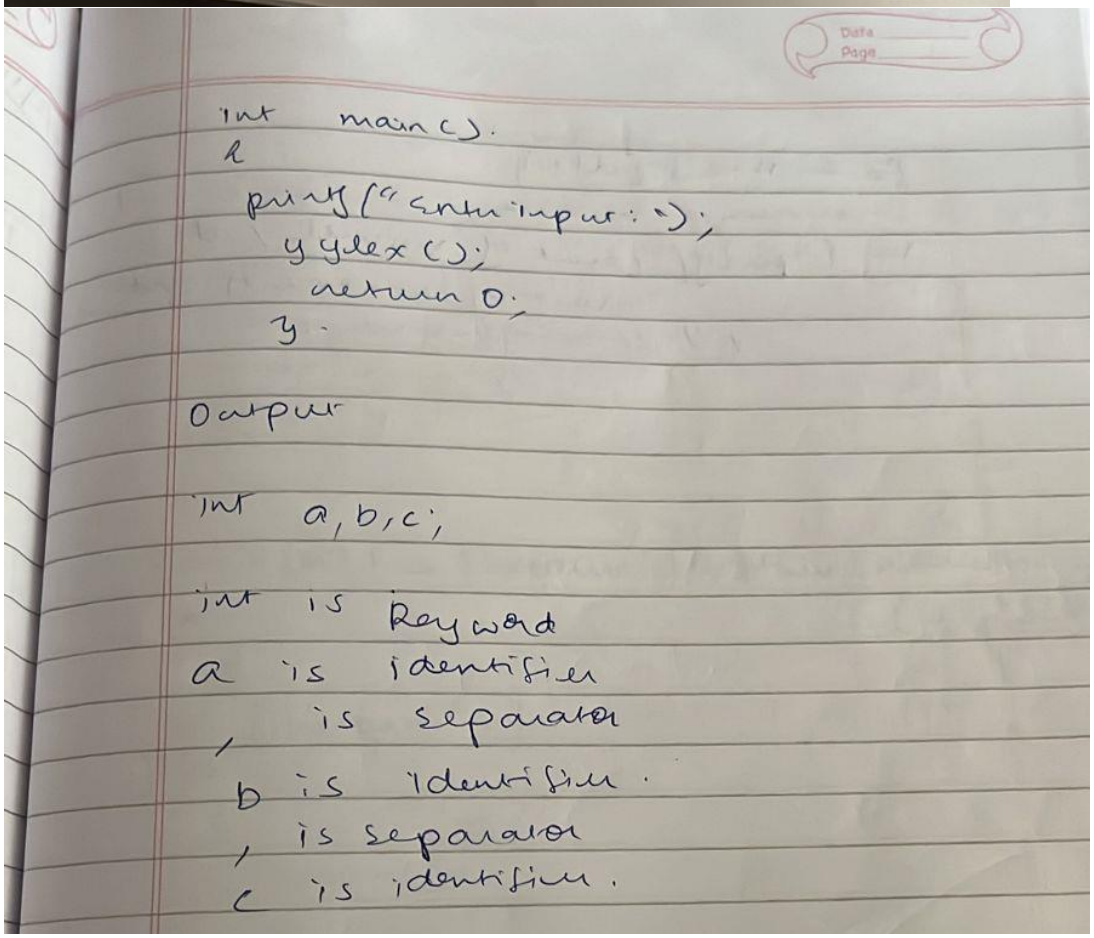
Output

```
Enter a 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
```

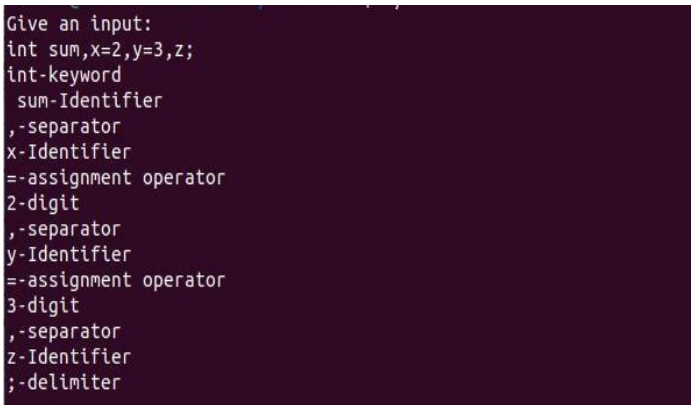
1.2

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



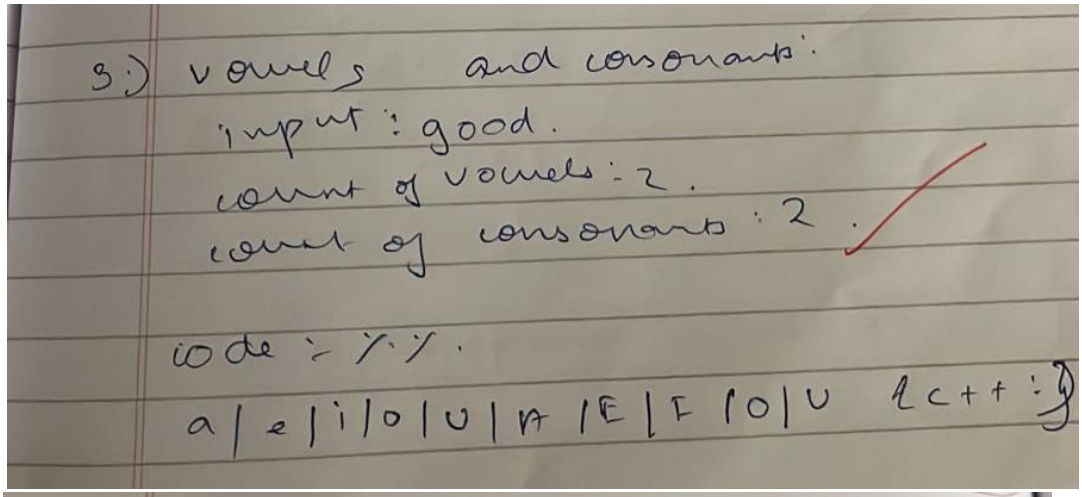


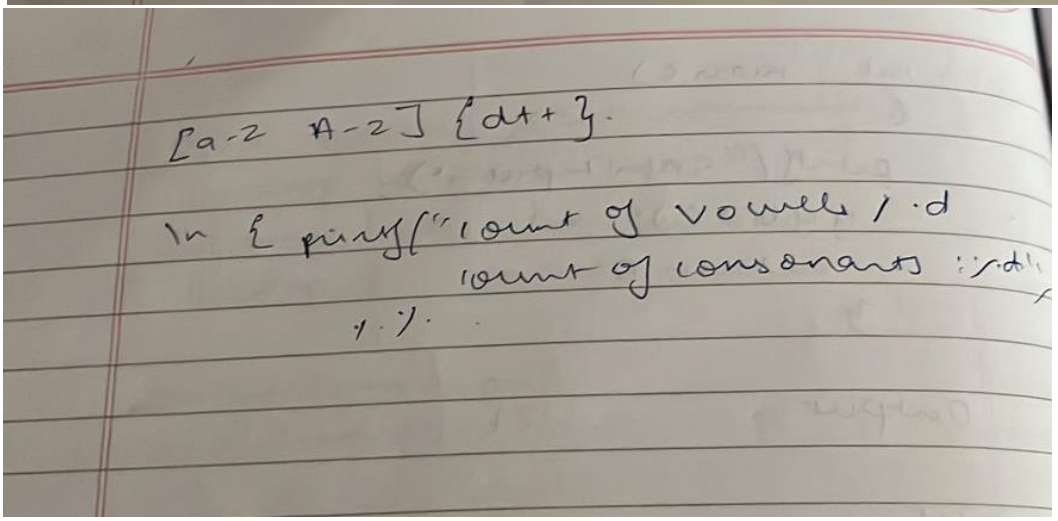
Output


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

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

Code


3.) vowels and consonants:
input: good.
count of vowels: 2.
count of consonants: 2.
code: %%.
a|e|i|o|u|A|E|I|O|U {c++;}


[a-z A-Z] {c++;}.
In { printf("count of vowels / .d
count of consonants : %d\n",
%%.

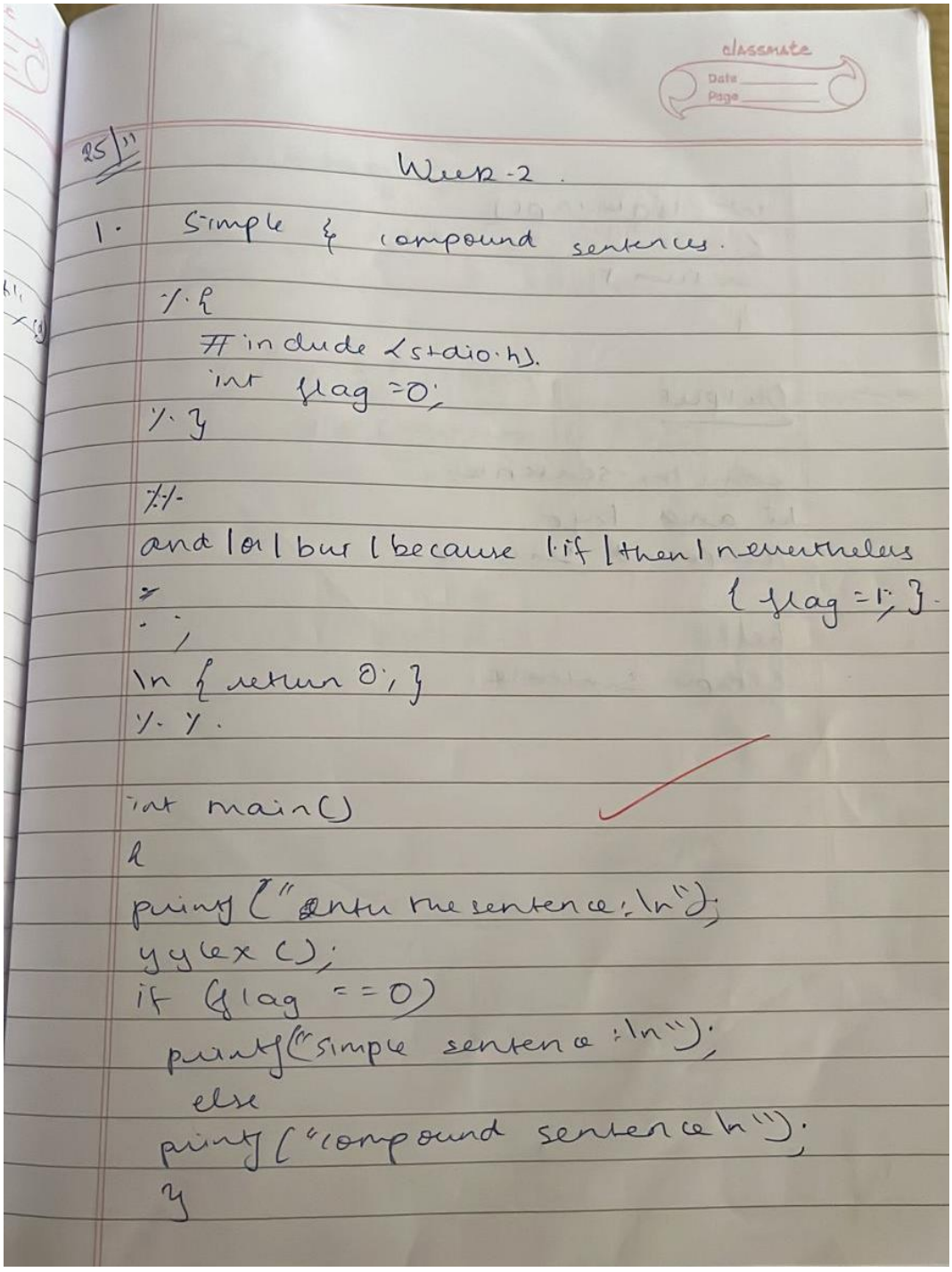
Output

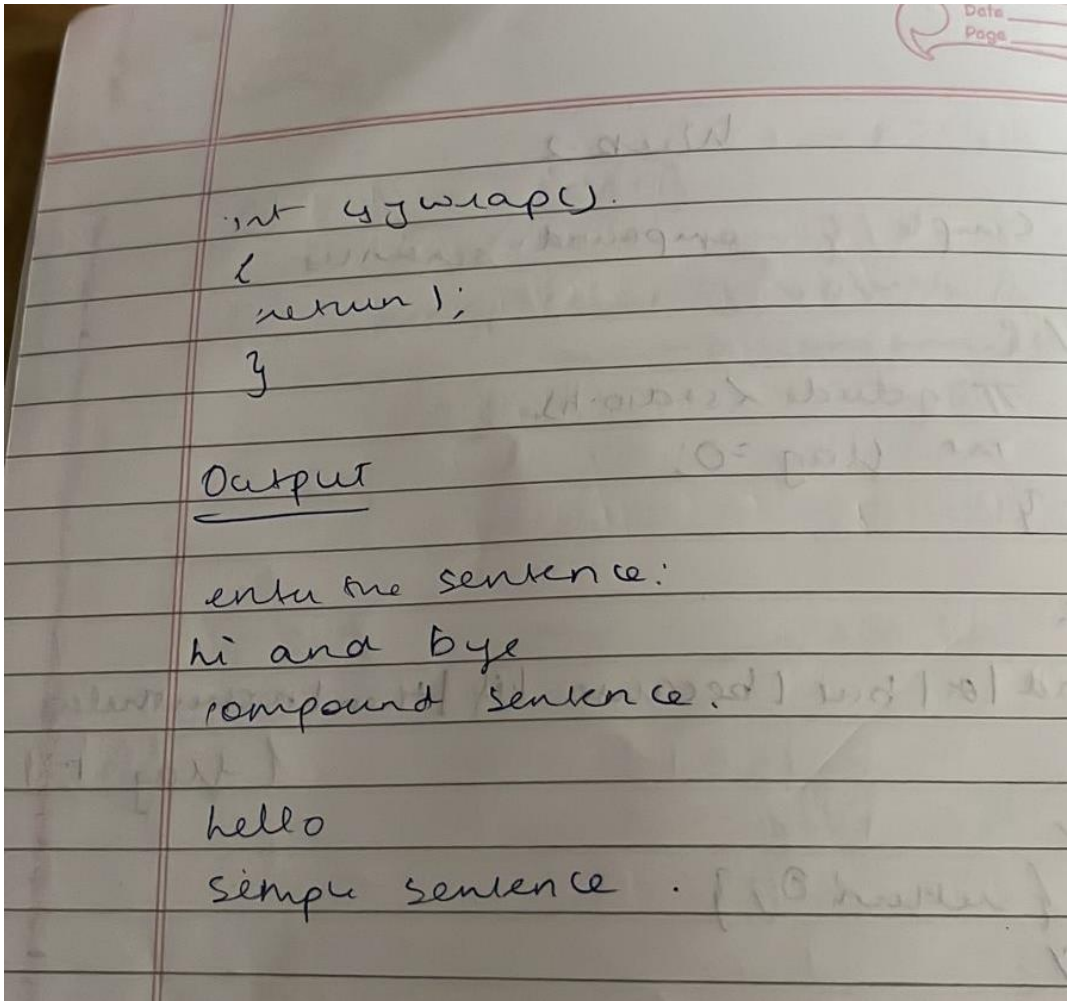
```
Enter a sentence:
Compiler design
 No of vowels and consonants are 5 and 9
This is a book
   No of vowels and consonants are 5 and 6
^C
```

Lab 2

2.1

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.

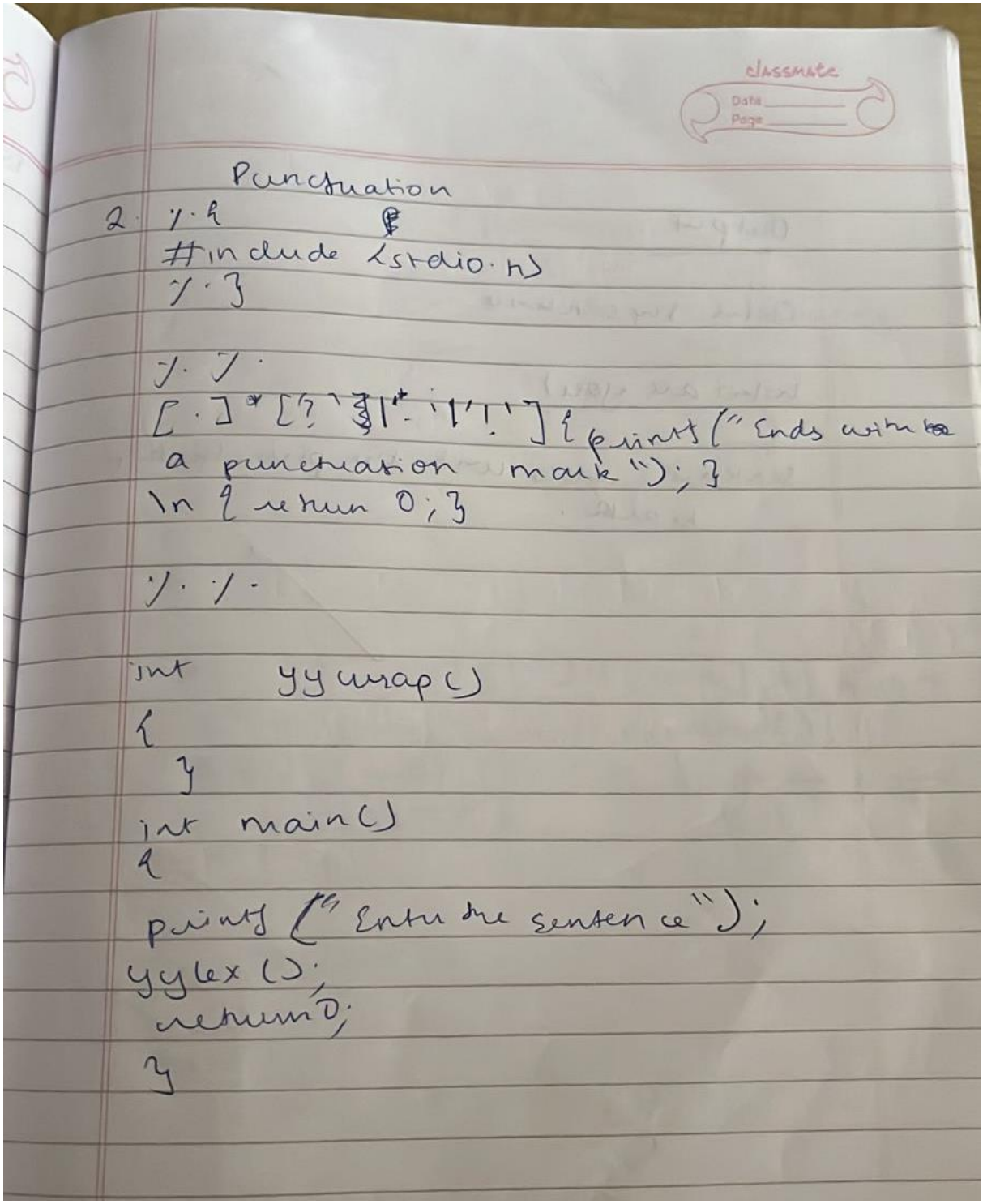




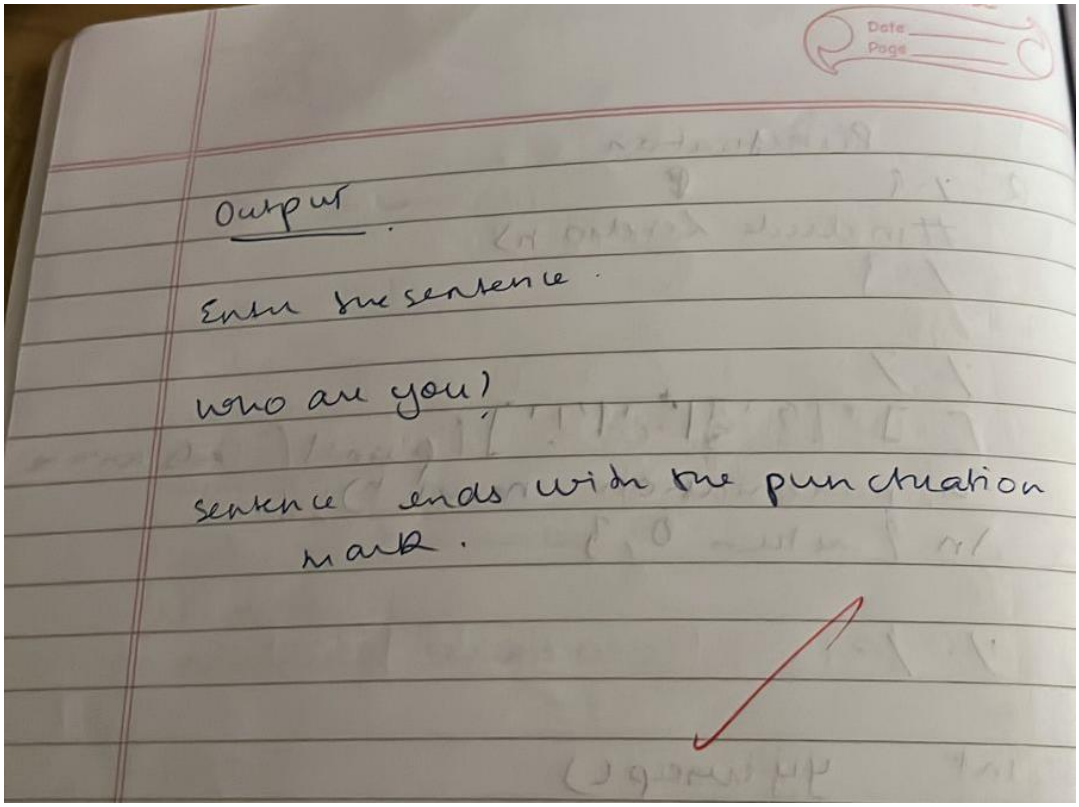
Output

```
Enter a sentence:
This is a car.
Simple sentence!
```

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



```
Punctuation
2. %{
#include <stdio.h>
%}

%%
[.]*[?|.|!] { printf("Ends with a punctuation mark"); }
\n { return 0; }

%%

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



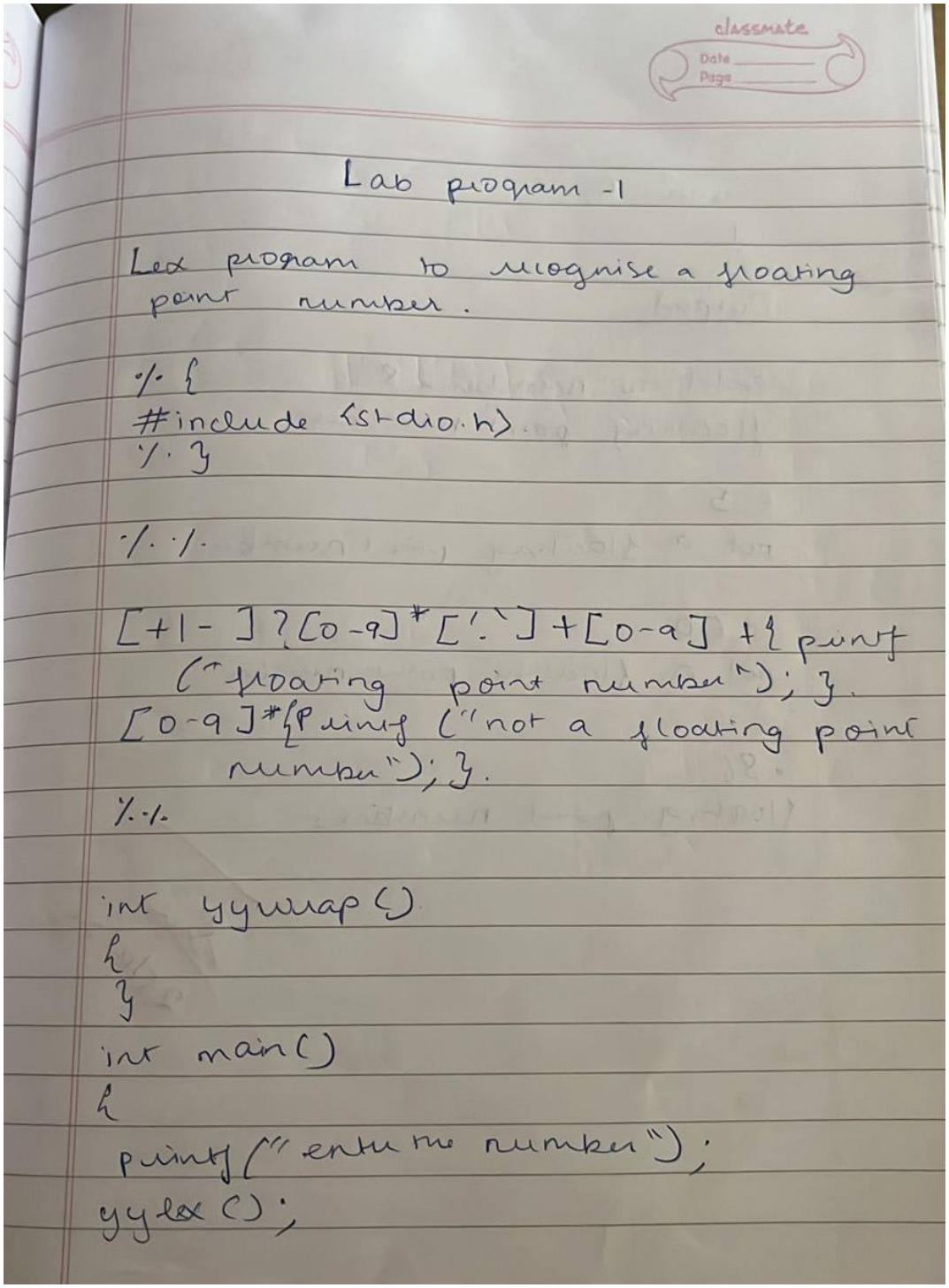
Output

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

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

Lab 3

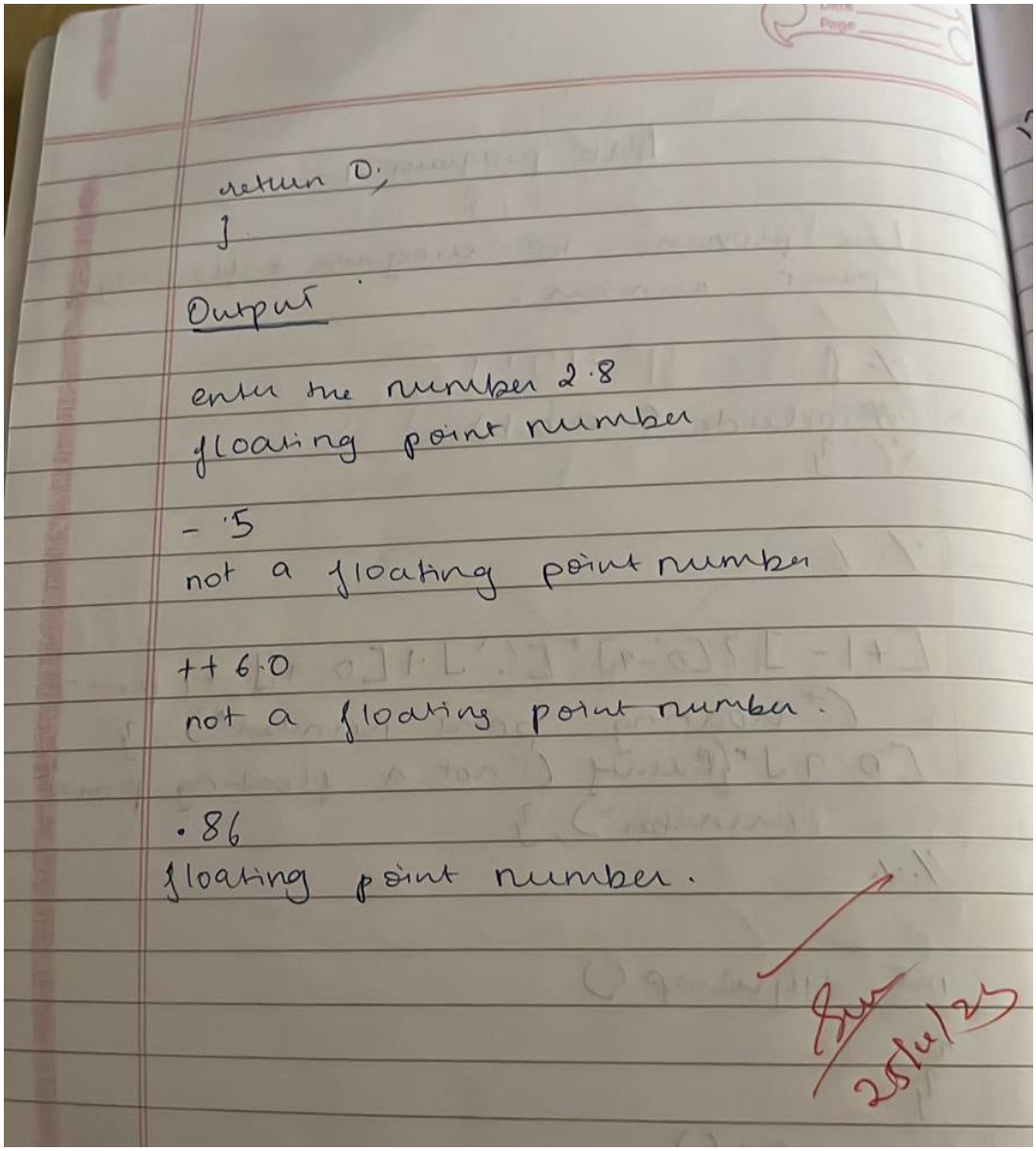
3.1 Write a program in LEX to recognize Floating Point Numbers.



Lab program -1

Lex program to recognise a floating point number.

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

%%

[+|-]?[0-9]*['.']+[0-9]+{printf("floating point number");}.
[0-9]*{printf("not a floating point number");}.

%%

int yywrap()
{
}
int main()
{
printf("enter the number");
yylex();
```

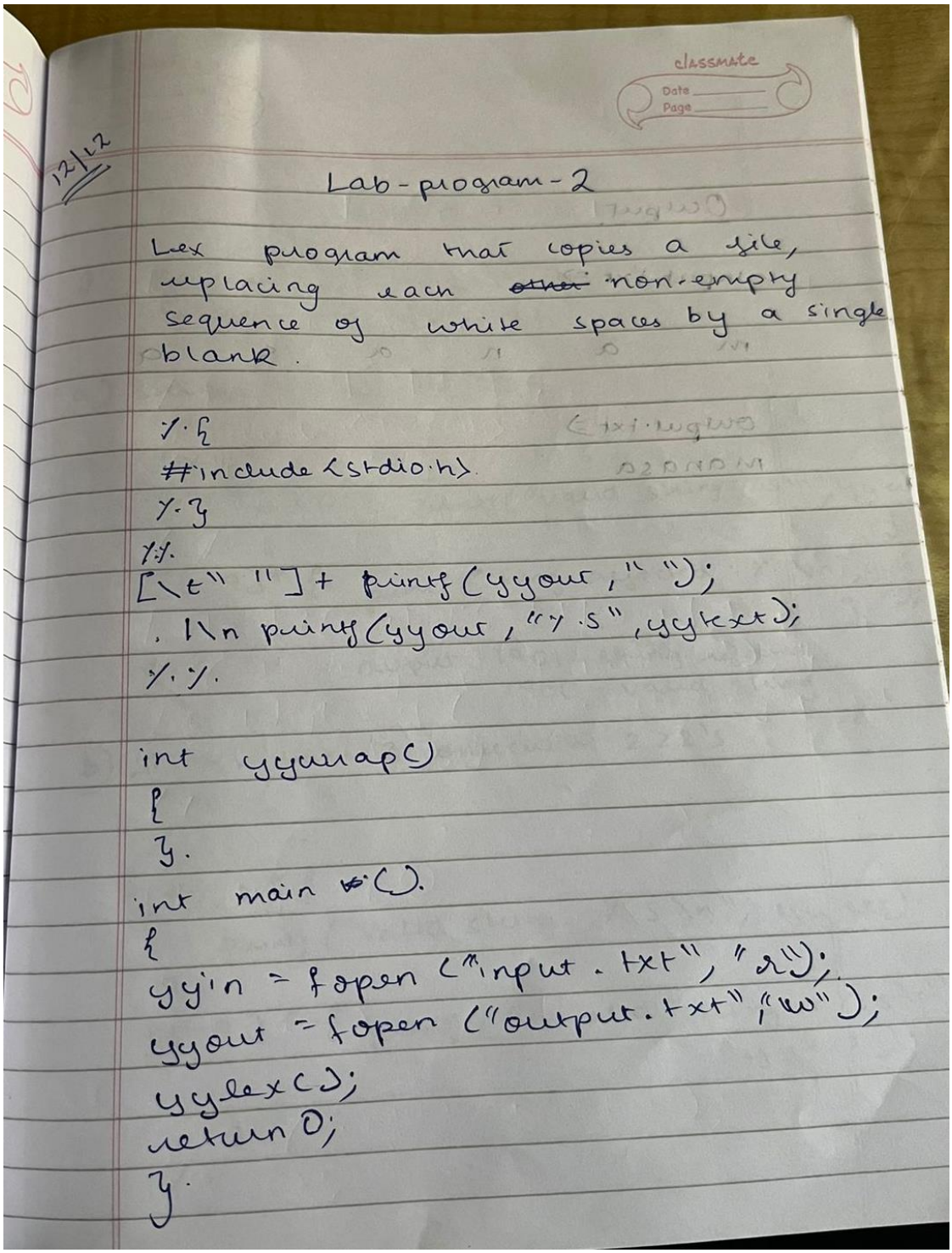


Output

```
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!
```

Lab 4

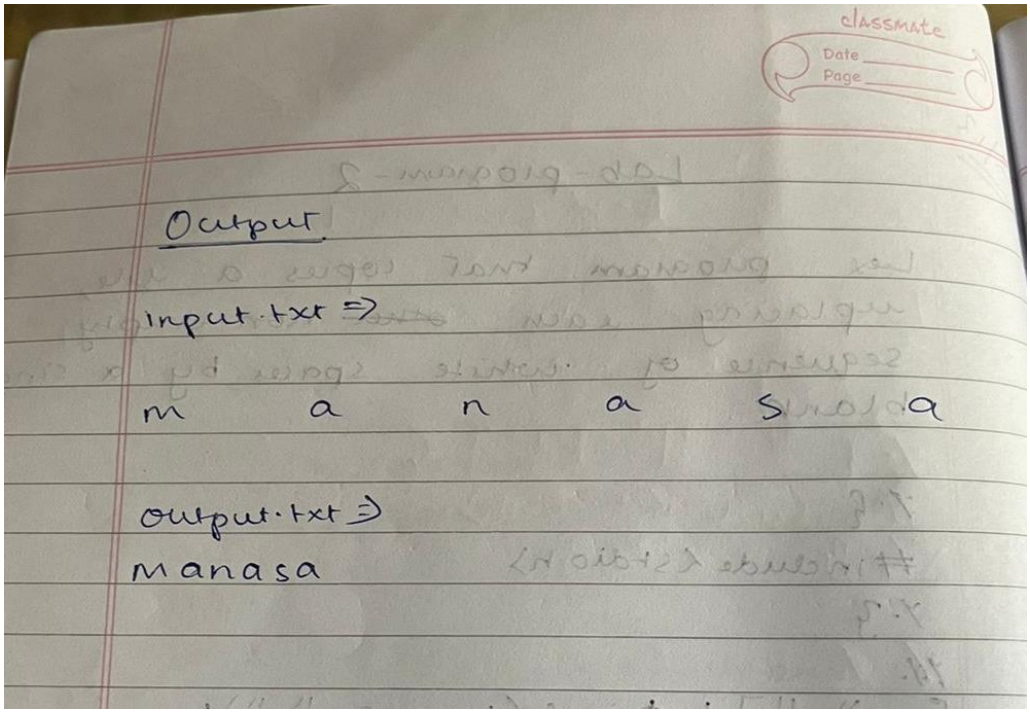
4.1 Write a LEX program that copies a file, replacing each nonempty sequence of white spaces by a single blank.



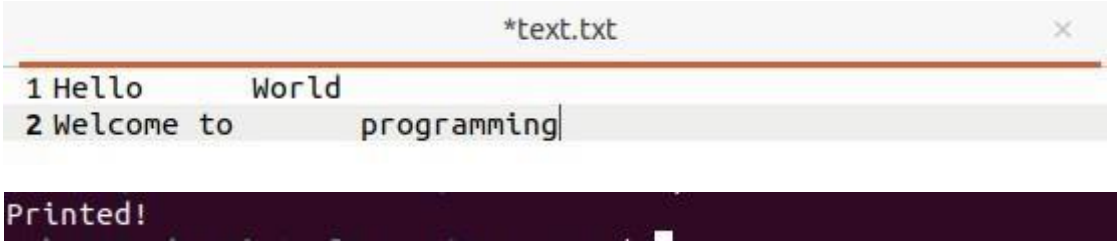
Lab - program - 2

Lex program that copies a file, replacing each ~~other~~ non-empty sequence of white spaces by a single blank.

```
%{
#include<stdio.h>
%}
%%
[\t" "]+ printf(yyout," ");
.|\n printf(yyout,"%s",yytext);
%%

int yywrap()
{
}
int main()
{
yyin = fopen("input.txt","r");
yyout = fopen("output.txt","w");
yylex();
return 0;
}
```


classmate
Date
Page
Output
input.txt ⇒
m a n a s
output.txt ⇒
manasa

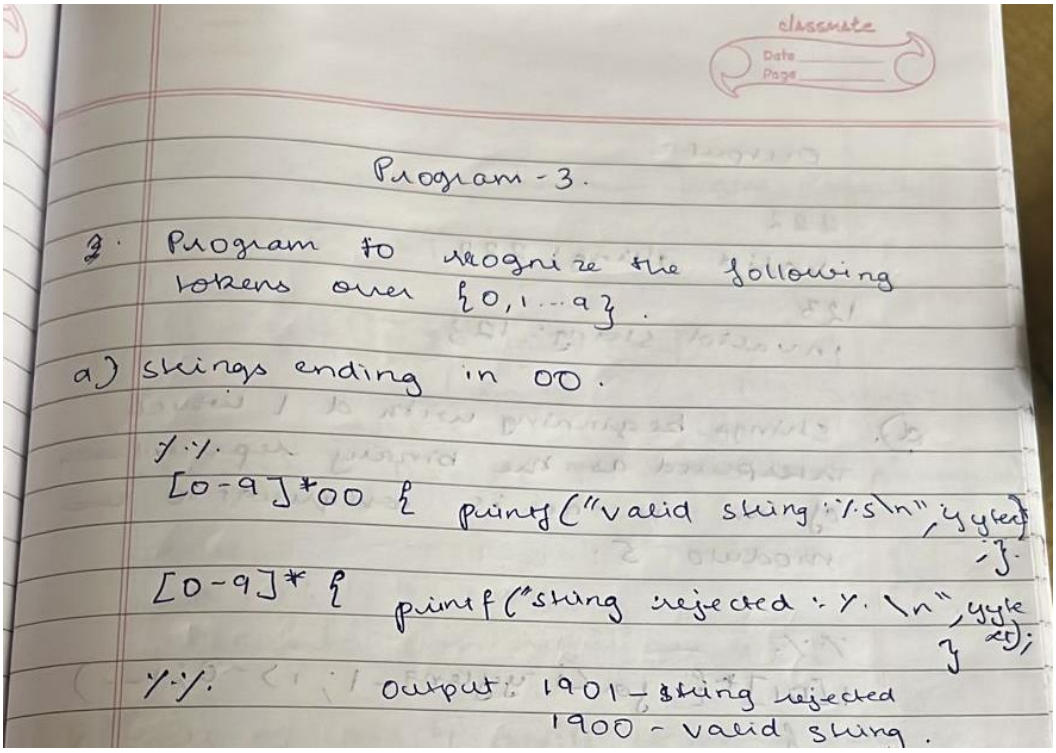
Output


*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.



Program - 3.

3. Program to recognize the following tokens over {0,1...9}.

a) strings ending in 00.

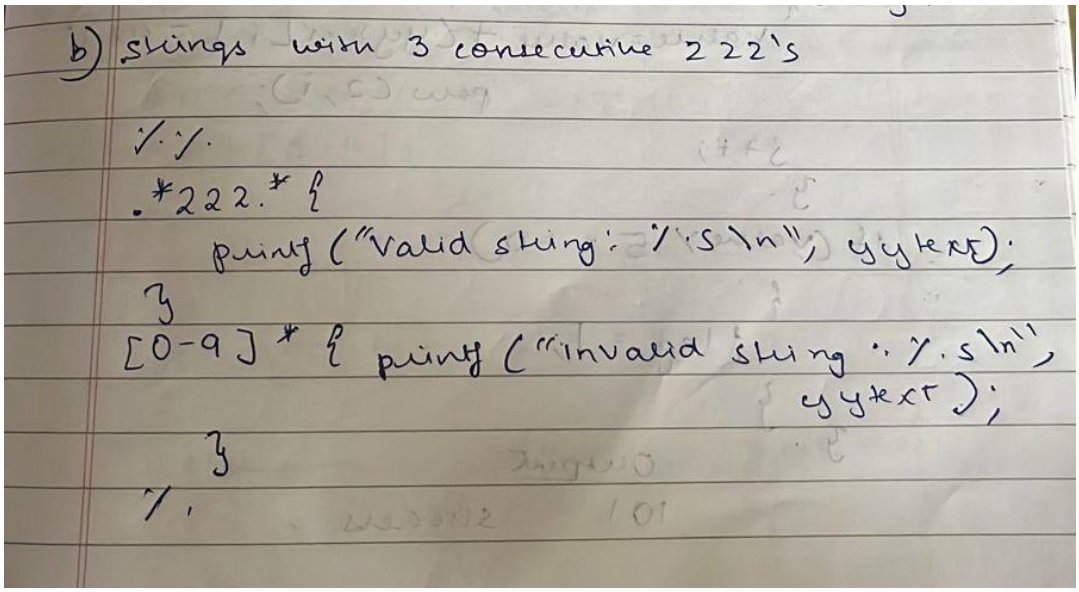
```
%%
[0-9]*00 { printf("valid string : %s\n", yytext);
[0-9]* { printf("string rejected : %s\n", yytext);
%%
```

output: 1901 - string rejected
1900 - valid string.

Output

```
Enter a string:
12300
Ends with 0.
```

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



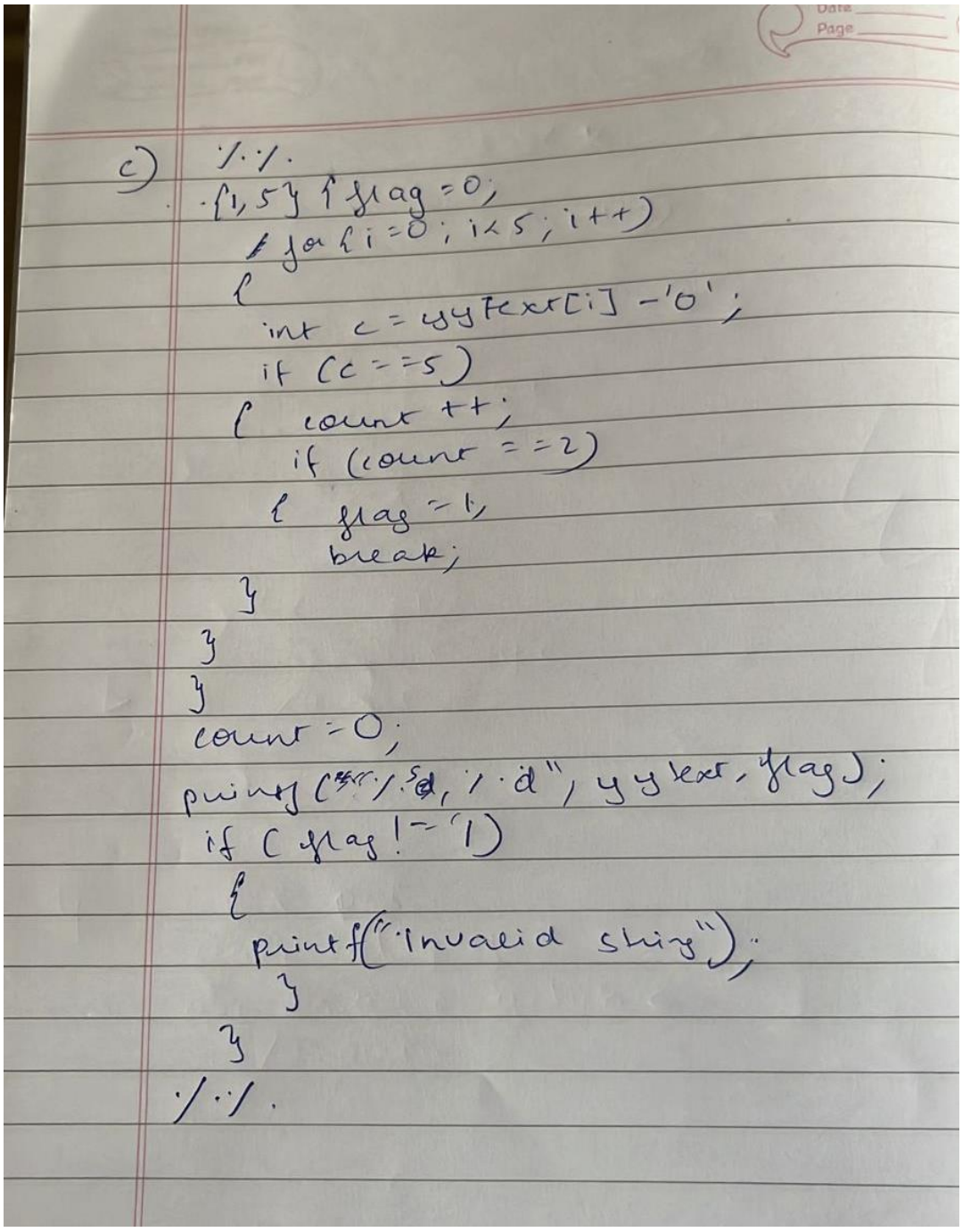
b) strings with 3 consecutive 222's

```
%%
.*222.* {
    printf("Valid string: %s\n", yytext);
}
[0-9]* { printf("invalid string : %s\n",
                yytext);
    }
%%
```

Output

```
Enter a string:
2322
Does not have 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.

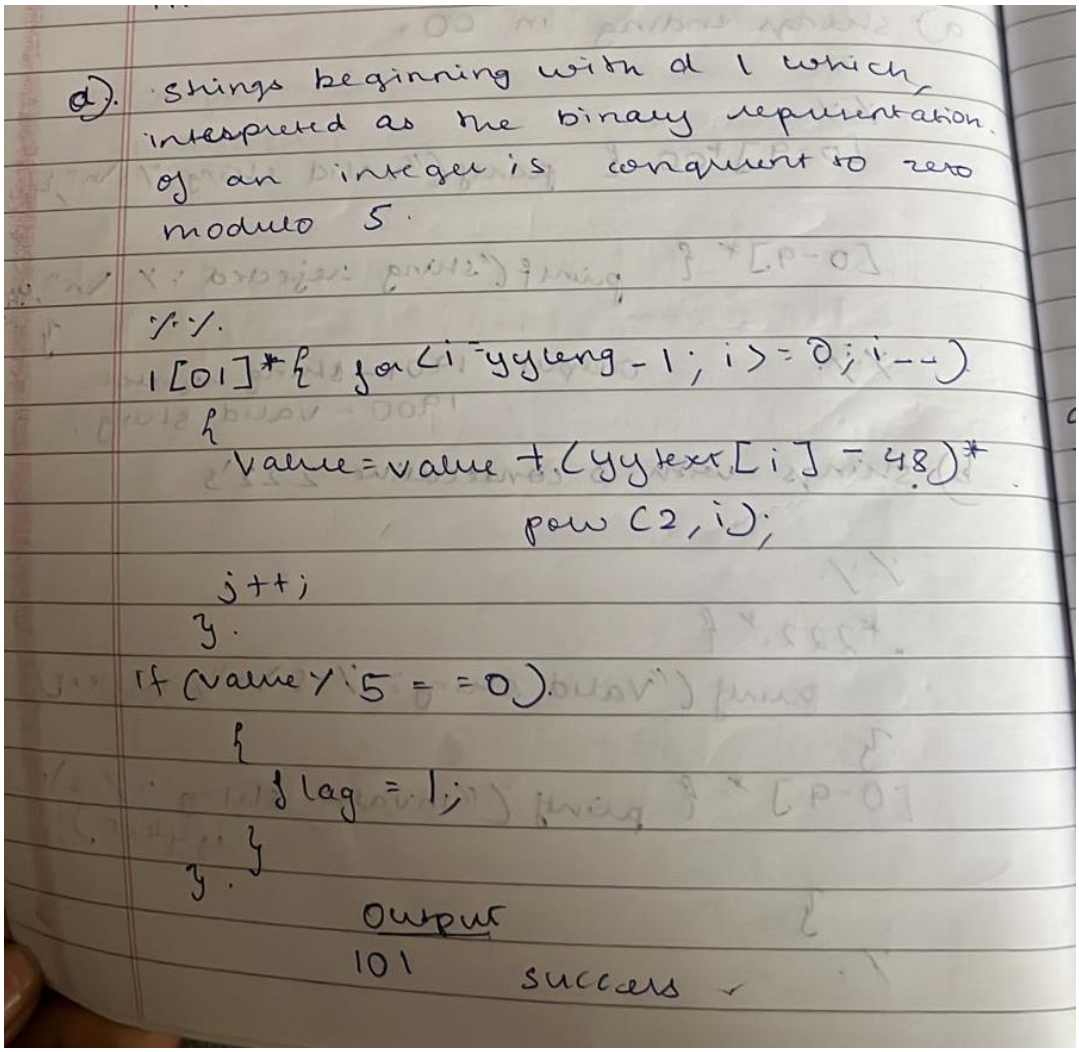


```
c) %:%.
[1,5] {flag=0;
  for(i=0;i<5;i++)
  {
   int c=yytext[i]-'0';
   if(c==5)
   { count++;
     if(count==2)
     { flag=1;
       break;
     }
   }
  }
  count=0;
  printf("%d,%d",yytext,flag);
  if(flag!='1')
   {
    printf("invalid string");
    }
  }
%:%.
```

Output

```
Enter a string:
1525558566
yytext:15255,flag(1 if no of 5 is atleast 2):1
yytext:58566,flag(1 if no of 5 is atleast 2):1
Valid string.
```

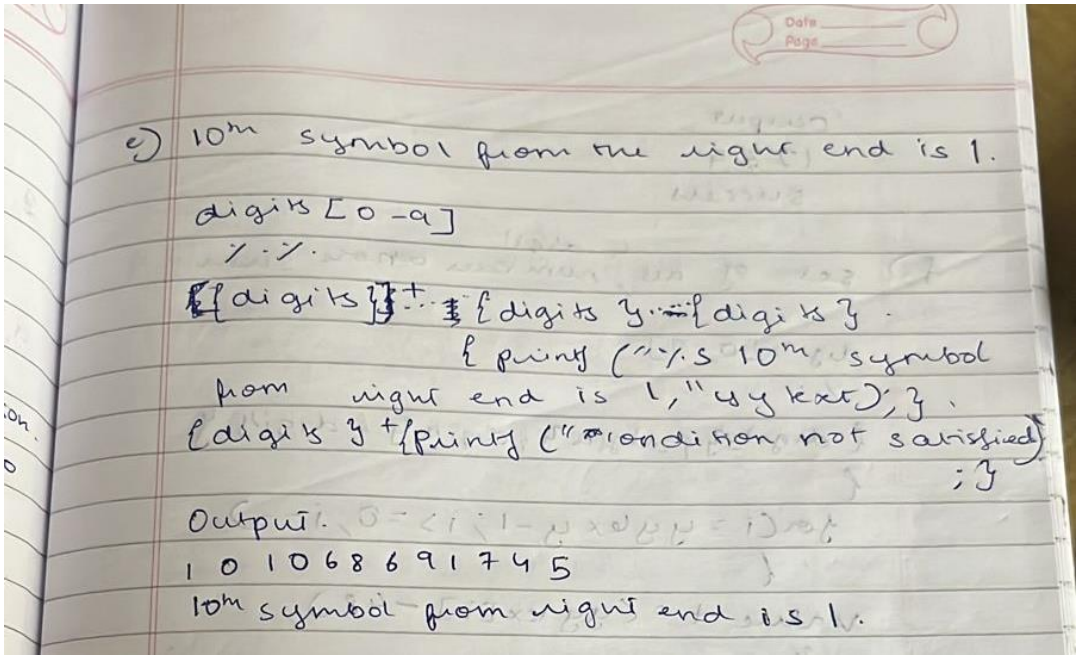
4.2.4 The set of all strings beginning with a 1 which, interpreted as the binary representation of an integer, is congruent to zero modulo 5.



Output

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

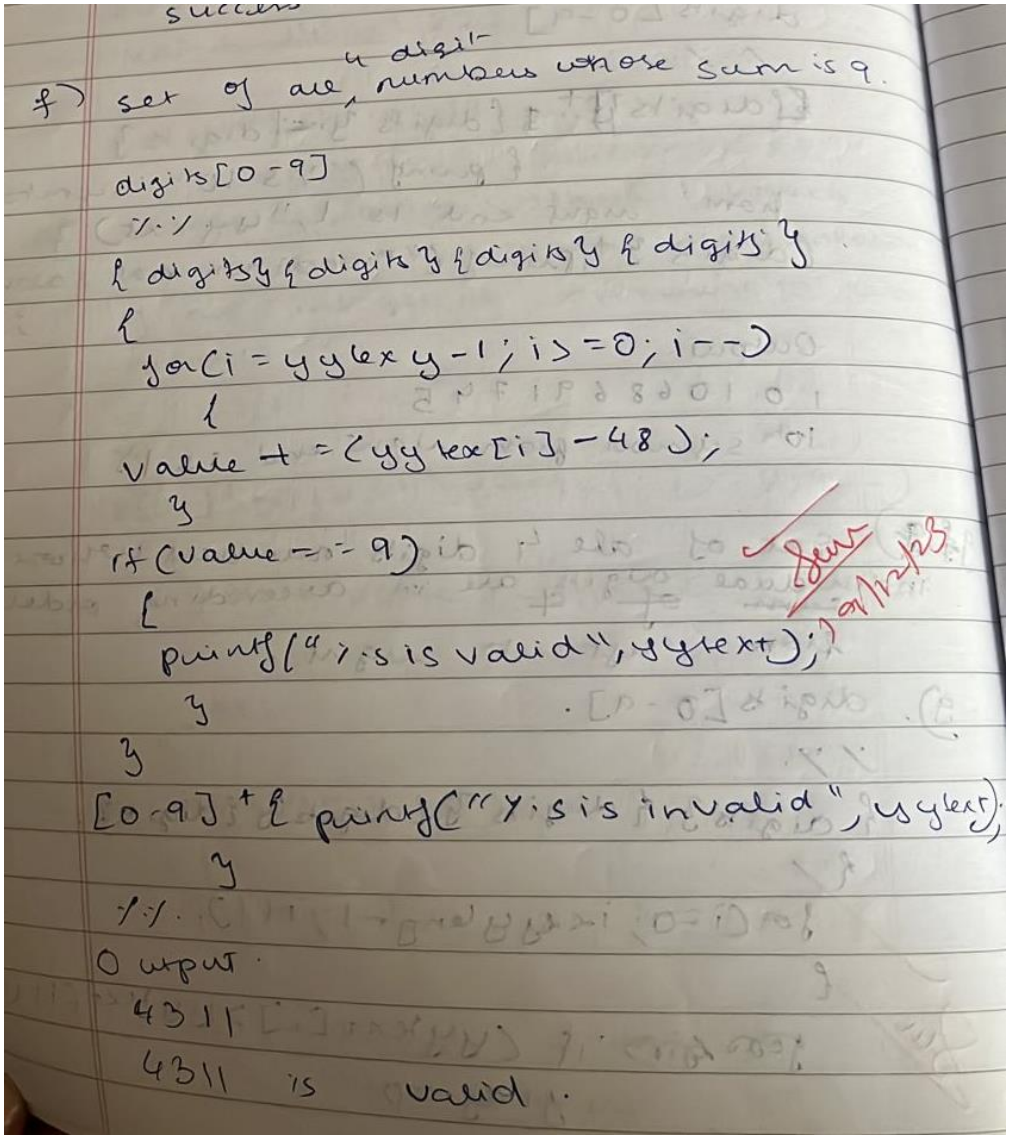
4.2.5 The set of all strings such that the 10th symbol from the right end is 1.



Output

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

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

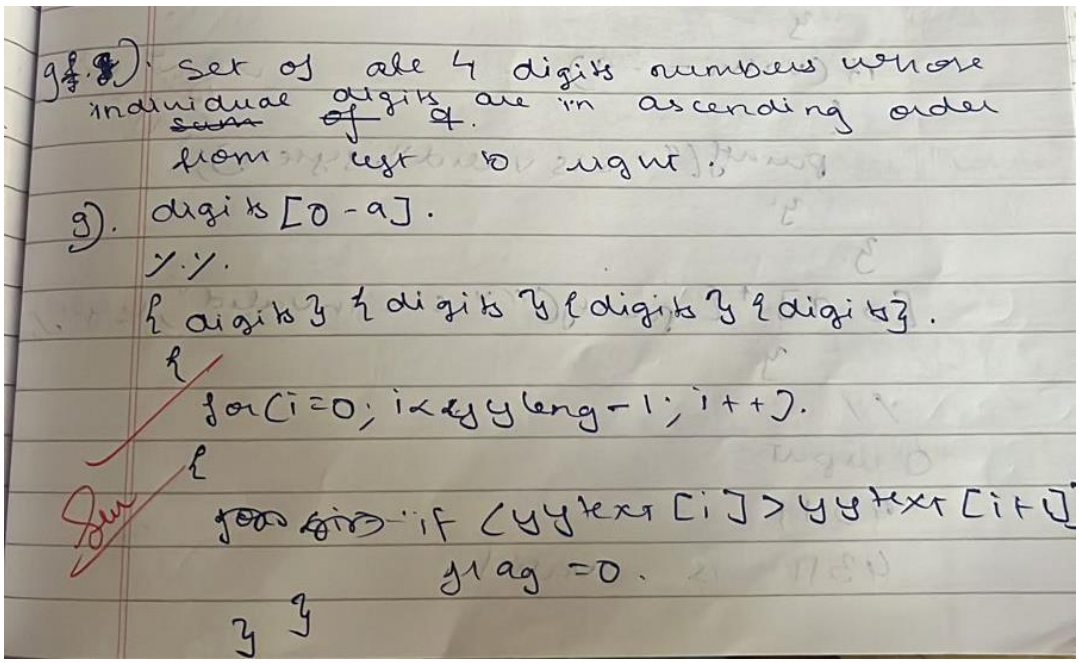


Output

```
Enter a string:
6300
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

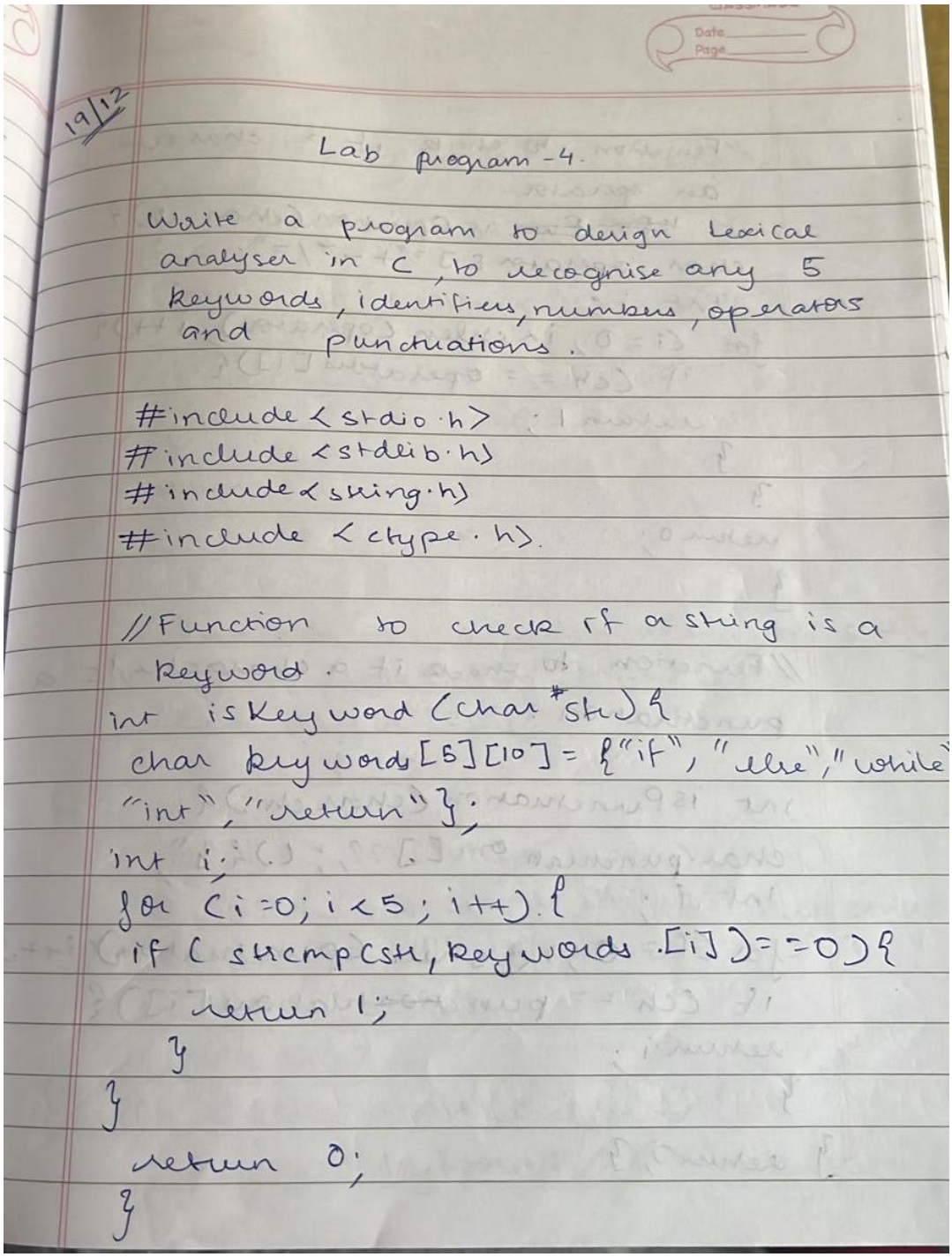


Output

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

Lab 5

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

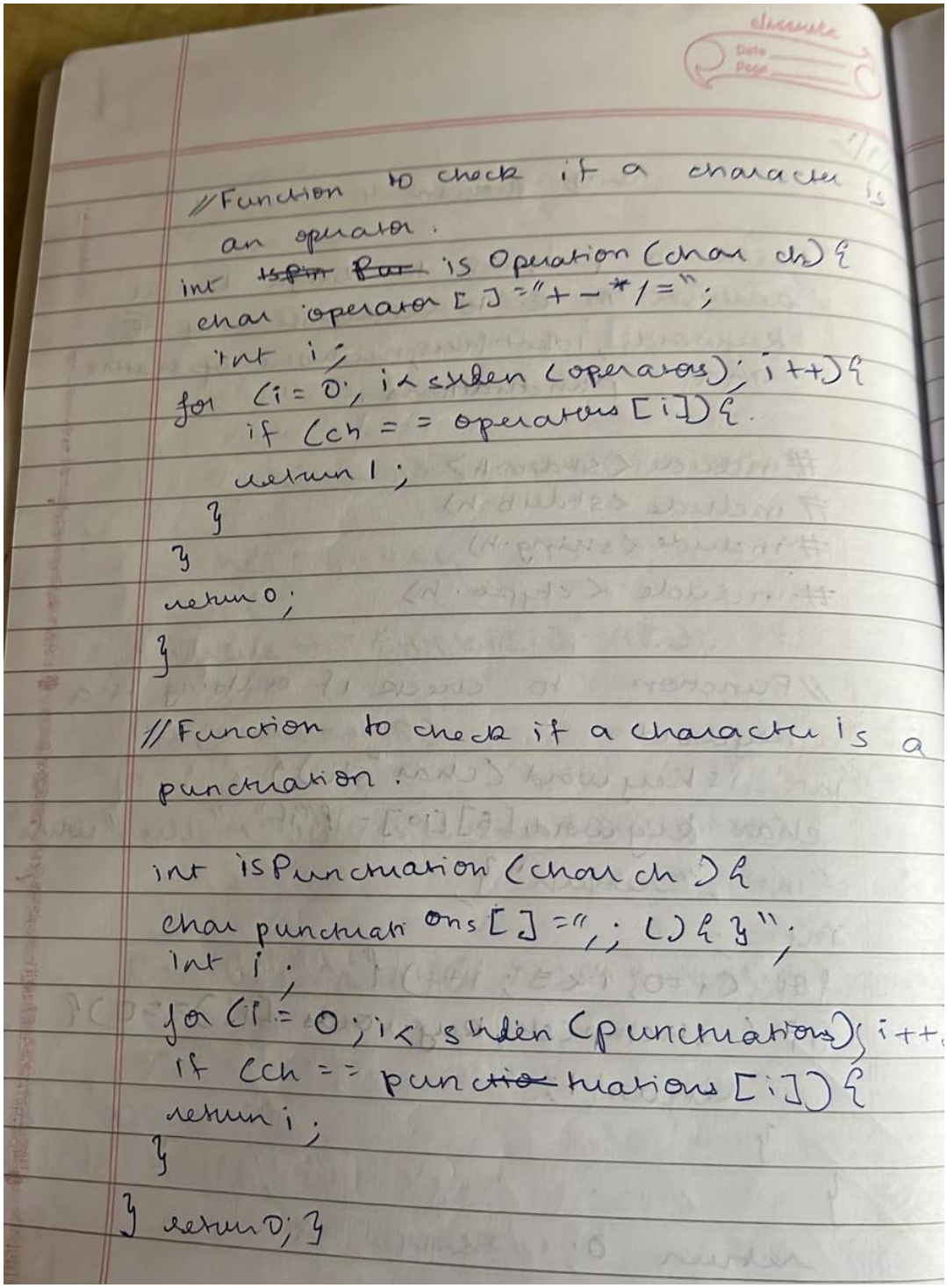


19/12

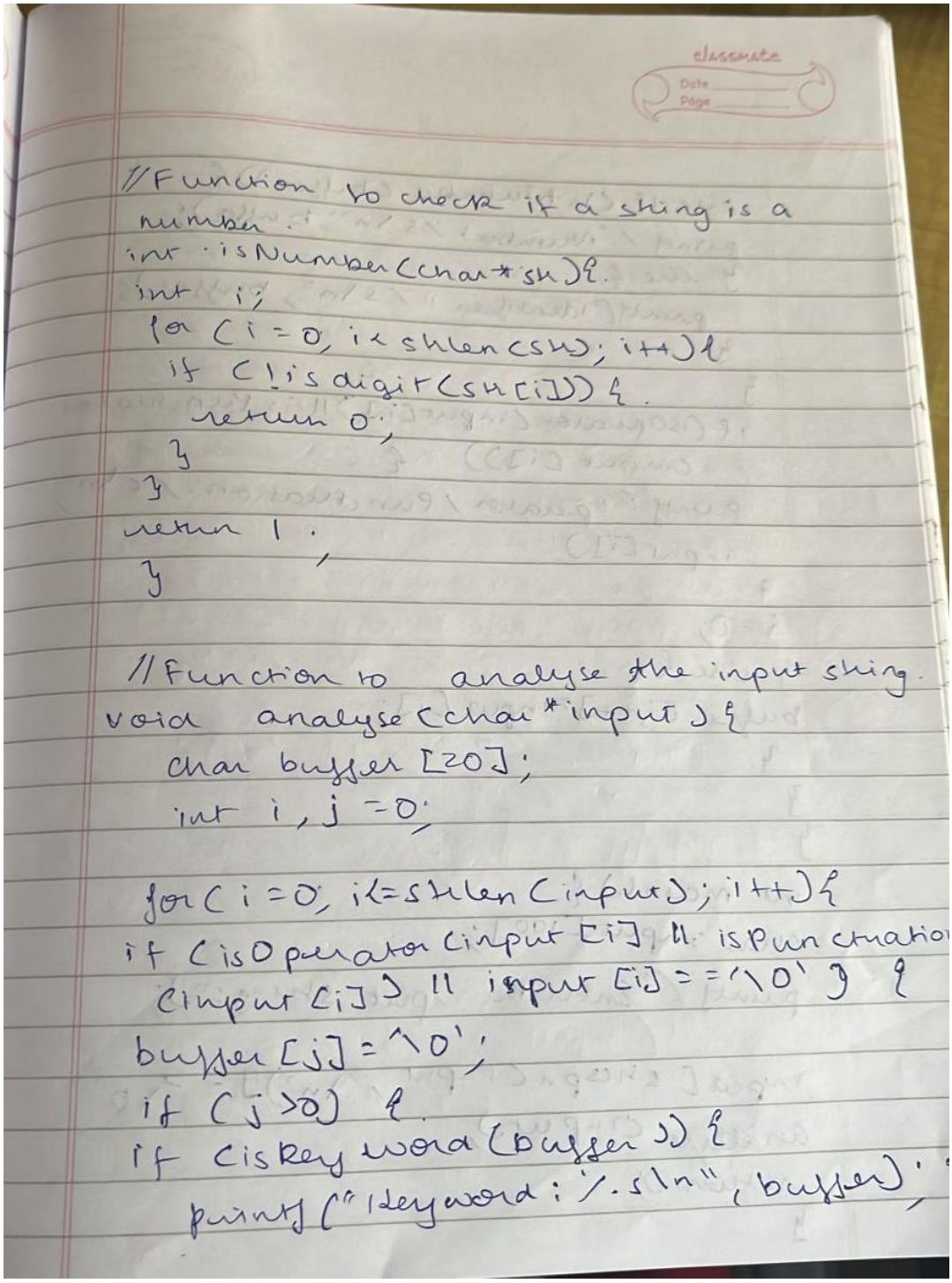
Lab program - 4.

Write a program to design Lexical analyser in C, to recognise any 5 Keywords, identifiers, numbers, operators and punctuations.

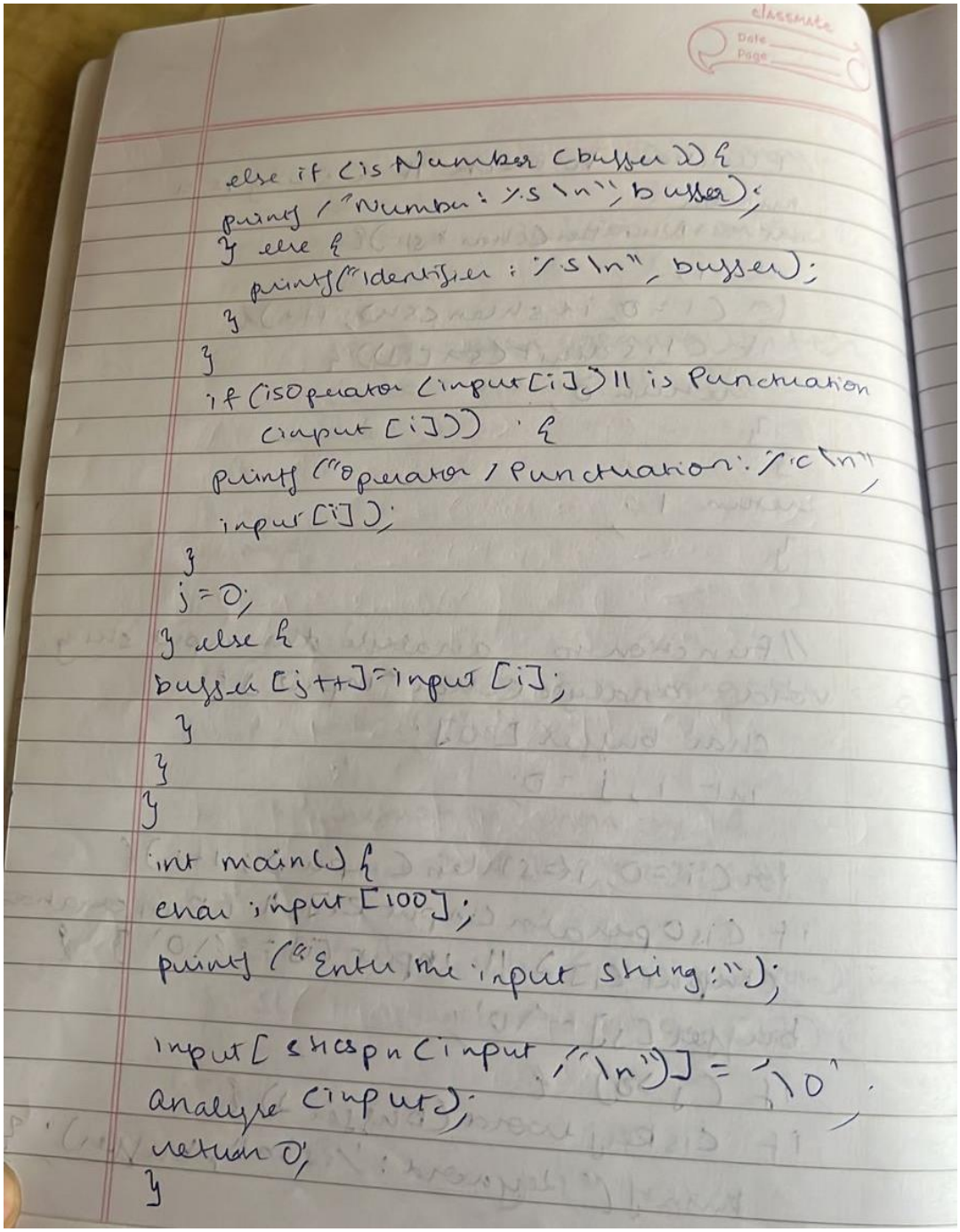
```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <ctype.h>

//Function to check if a string is a Keyword.
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;
}
```



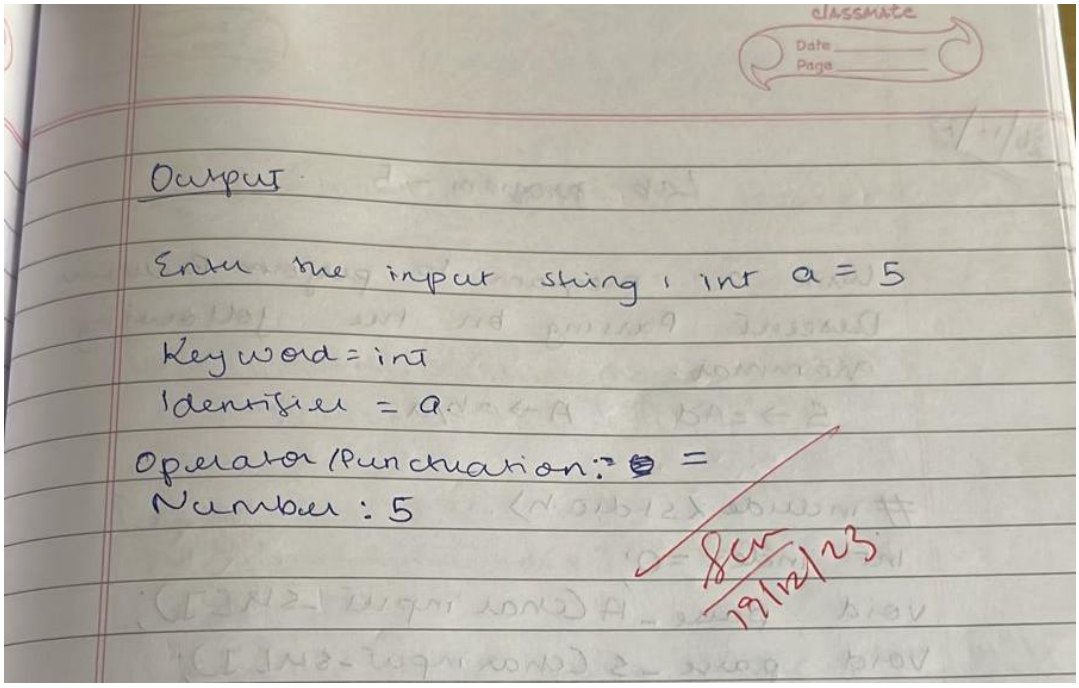
```
//Function to check if a character is
  an operator.
int ~~isPun~~ ~~Pun~~ isOperation(char ch){
 char operators[] = "+-*/=";
  int i;
for(i=0; i<sizeof(operators); i++){
   if(ch == operators[i]){
   return 1;
  }
}
return 0;
}

//Function to check if a character is a
 punctuation.

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



```
//Function to check if a string is a
number.
int isNumber(char *str){
int i;
for(i=0; i<strlen(str); i++){
  if(!isdigit(str[i])){
    return 0;
  }
}
return 1;
}

//Function to analyse the input string.
void analyse(char *input){
   char buffer[20];
   int i, j=0;

  for(i=0; i<=strlen(input); i++){
if(isOperator(input[i]) || isPunctuatio
  (input[i]) || input[i]=='\0'){
buffer[j]='\0';
if(j>0){
if(isKeyword(buffer)){
   printf("Keyword: %s\n", buffer);
```



```
else if (isNumber(buffer)) {
printf("Number: %s\n", buffer);
} else {
  printf("Identifier: %s\n", buffer);
}
}
if (isOperator(input[i]) || isPunctuation
    (input[i])) {
printf("Operator / Punctuation: %c\n",
 input[i]);
}
j=0;
} else {
buffer[j++]=input[i];
}
}
}
int main() {
char input[100];
printf("Enter the input string:");

input[strcspn(input, "\n")] = '\0';
analyse(input);
return 0;
}
```



Output

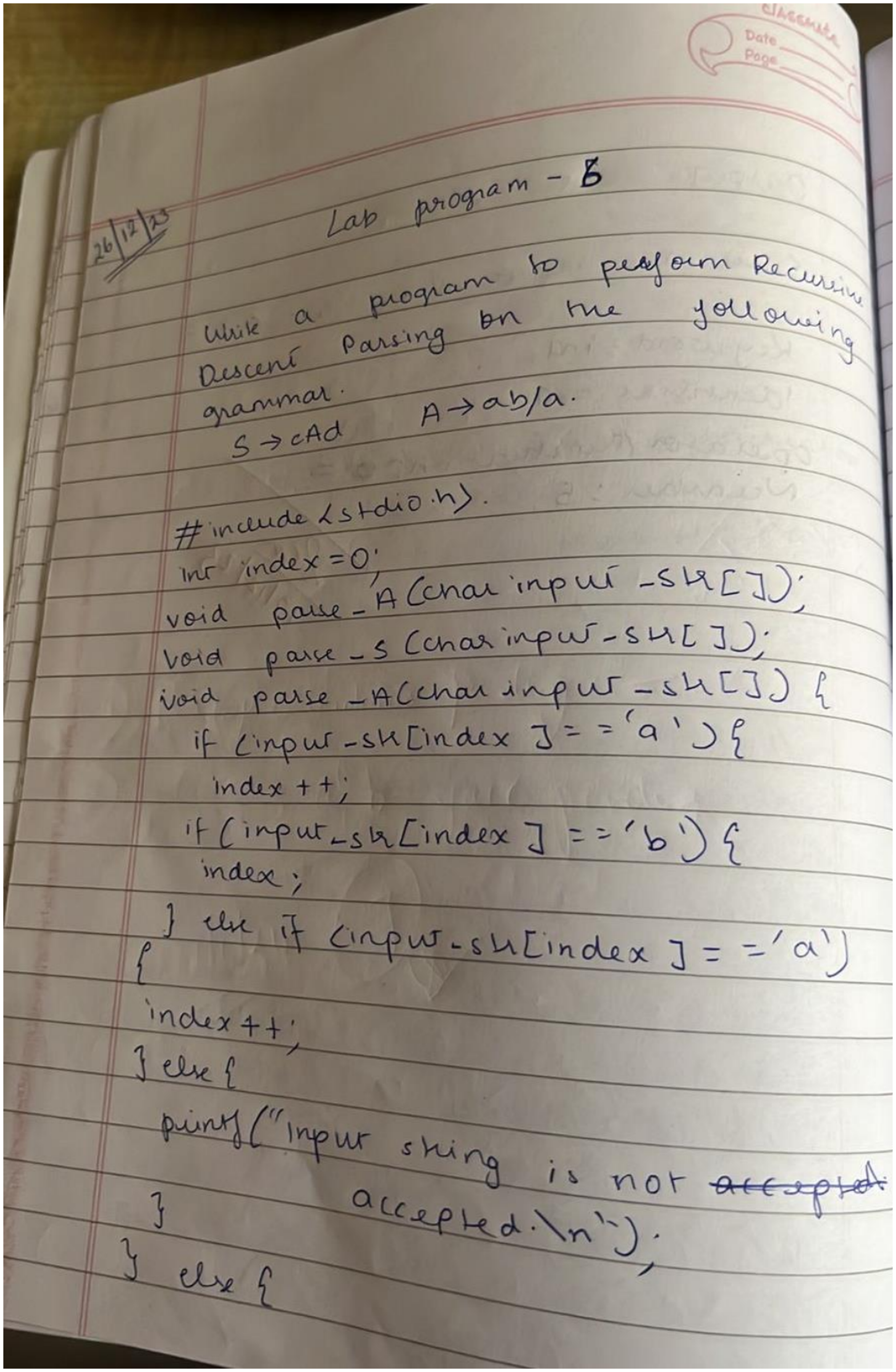
```
Keyword: if
Operator: (
Identifier: x
Operator: >
Number: 0
Operator: )
Operator: {
Keyword: return
Identifier: x
Punctuation:;
Operator: }
Keyword: else
Operator: {
Keyword: return
Operator: -x
Punctuation:;
Operator: }
```

Lab 6

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

S->cAd

A->ab | a



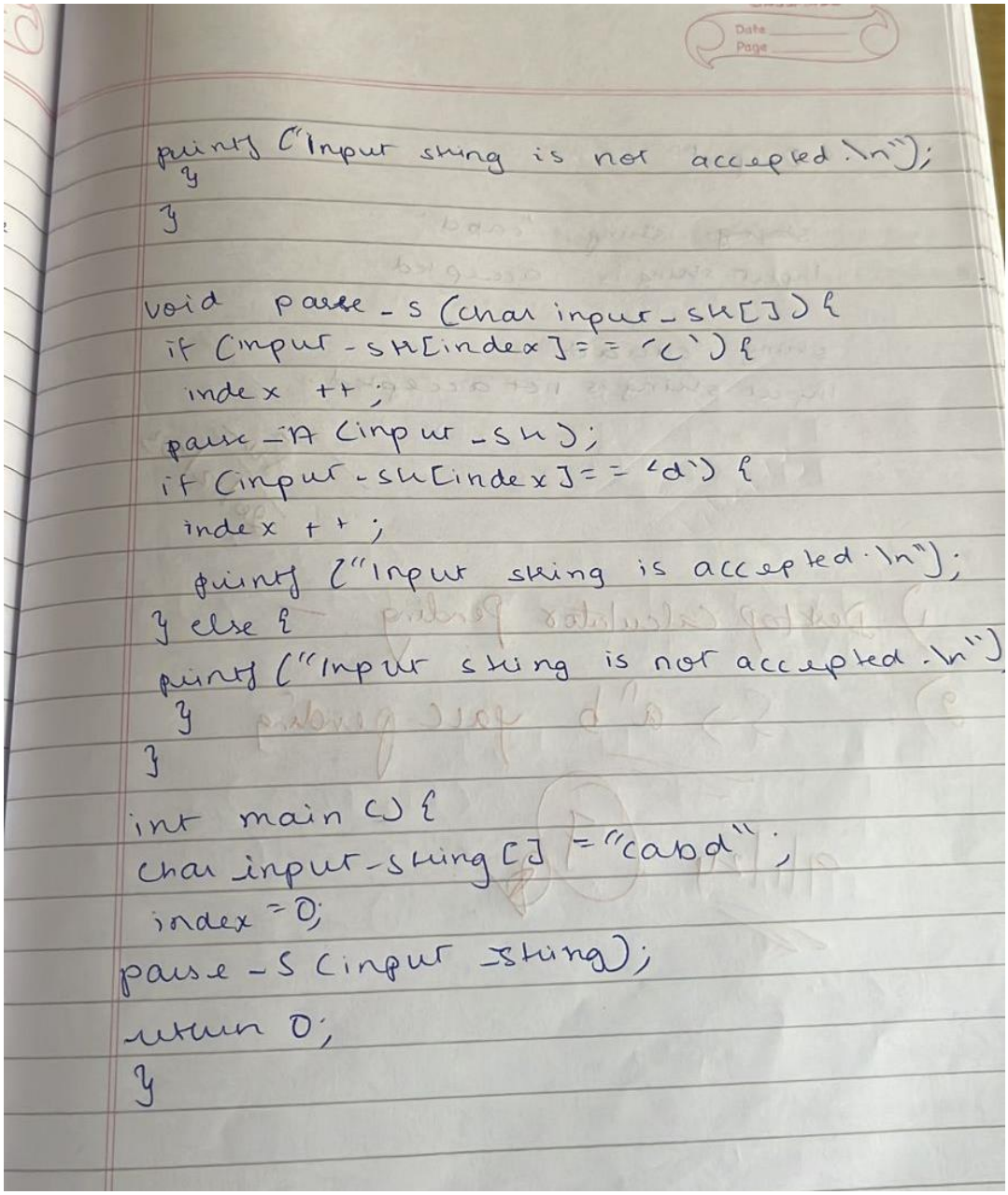
Lab program - 6

26/12/25

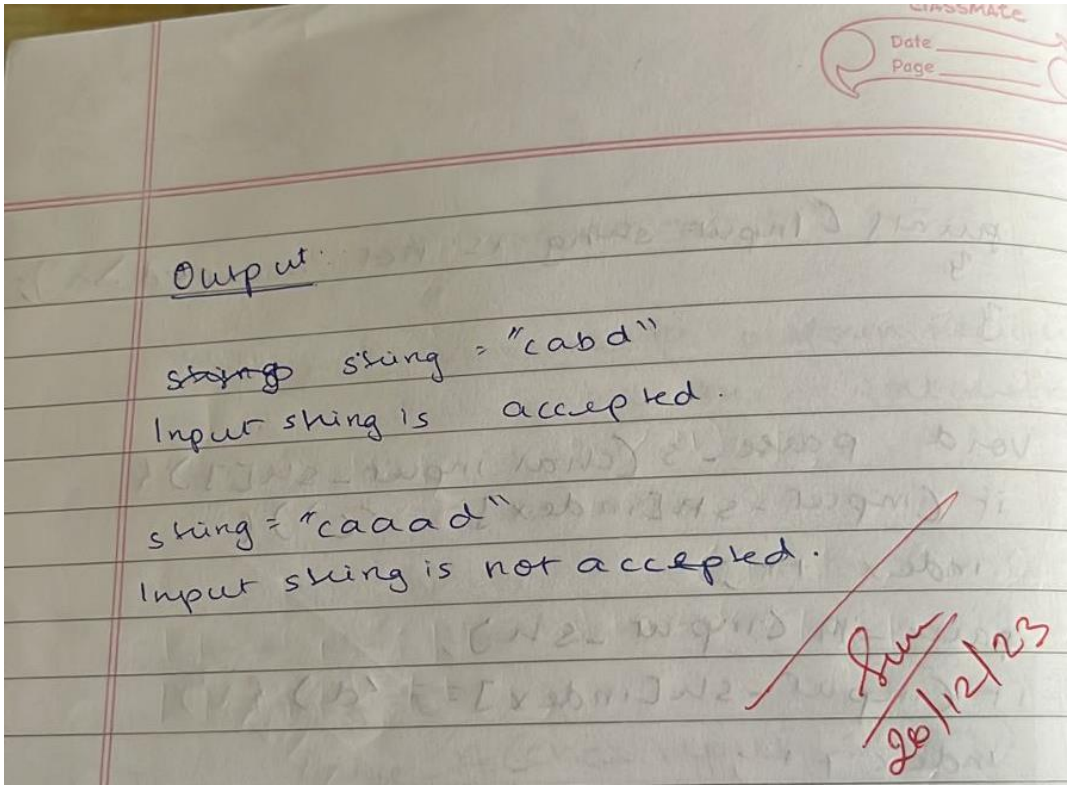
Write a program to perform Recursive Descent Parsing on the following grammar.

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.
          accepted.\n");
  }
} else{
```



```
printf("Input string is not accepted.\n");
 }
}

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.\n");
 }else{
 printf("Input string is not accepted.\n")
  }
}
int main(){
char input_string[] = "cabd";
 index=0;
parse_S(input_string);
return 0;
}
```

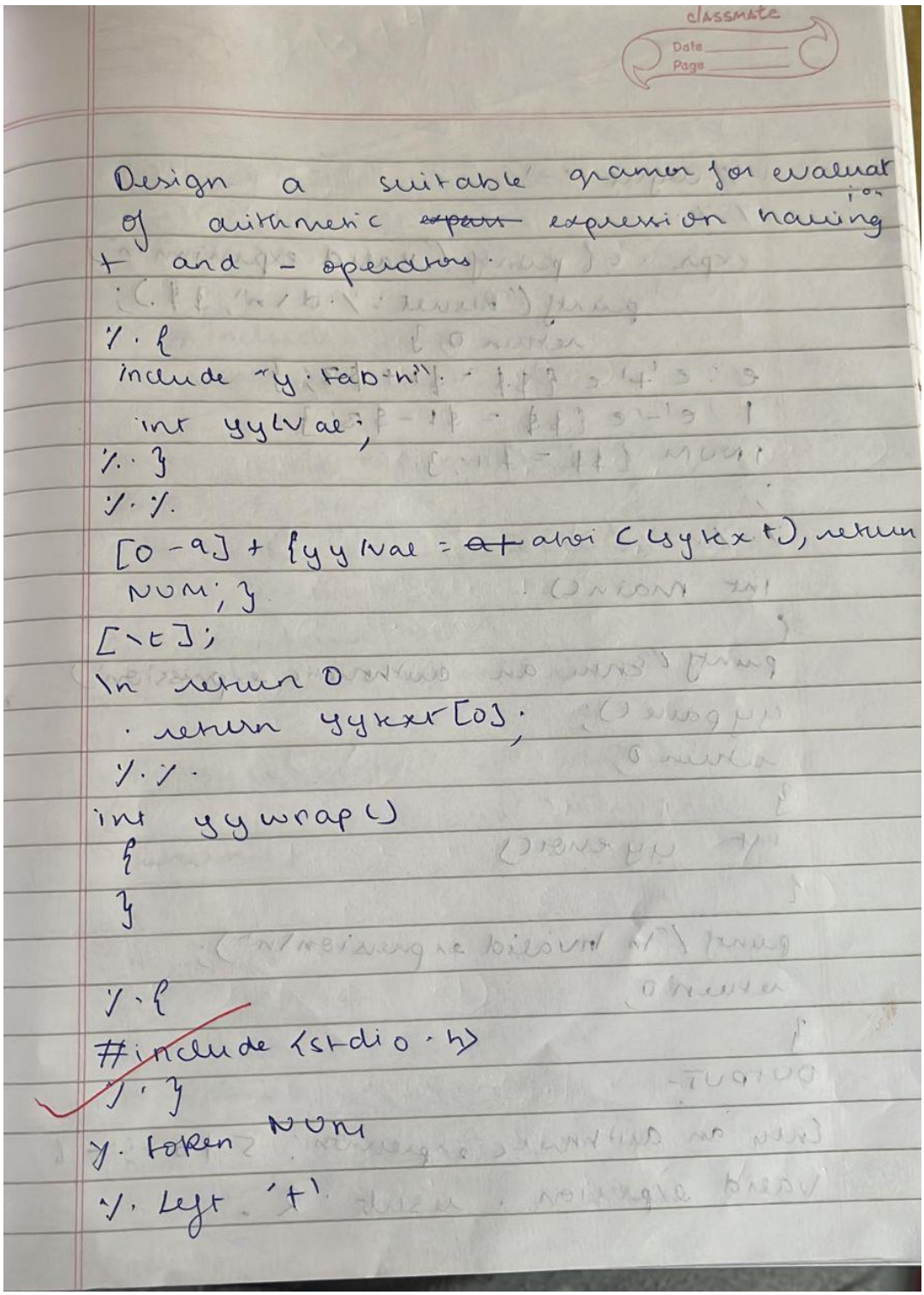


Output

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

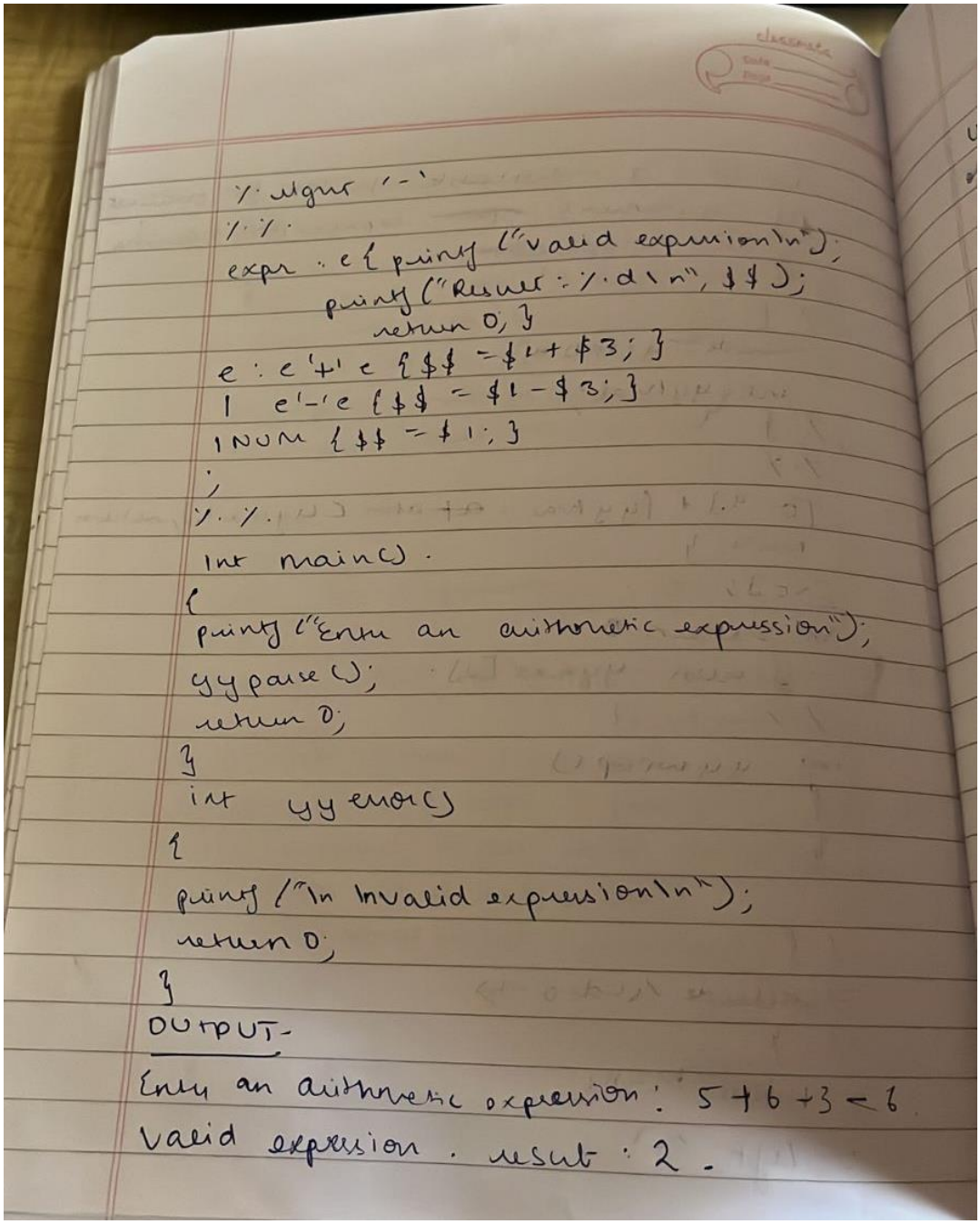
Lab 7

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



Design a suitable gramar for evaluat
of arithmetic ~~expres~~ expression having
+ and - operators.

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

%{
#include <stdio.h>
%}
%token NUM
%left '+'
```

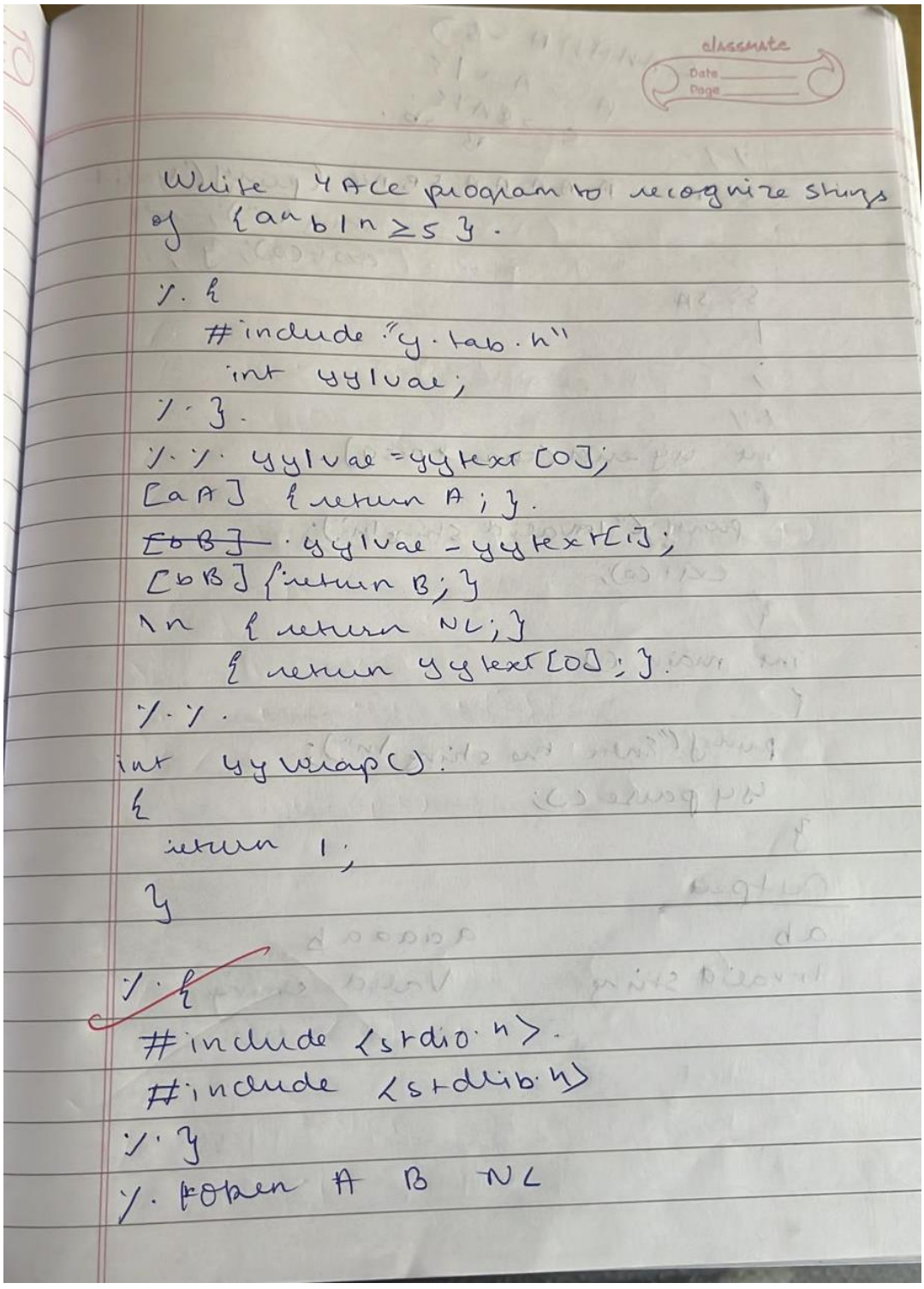


Output

```
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}.

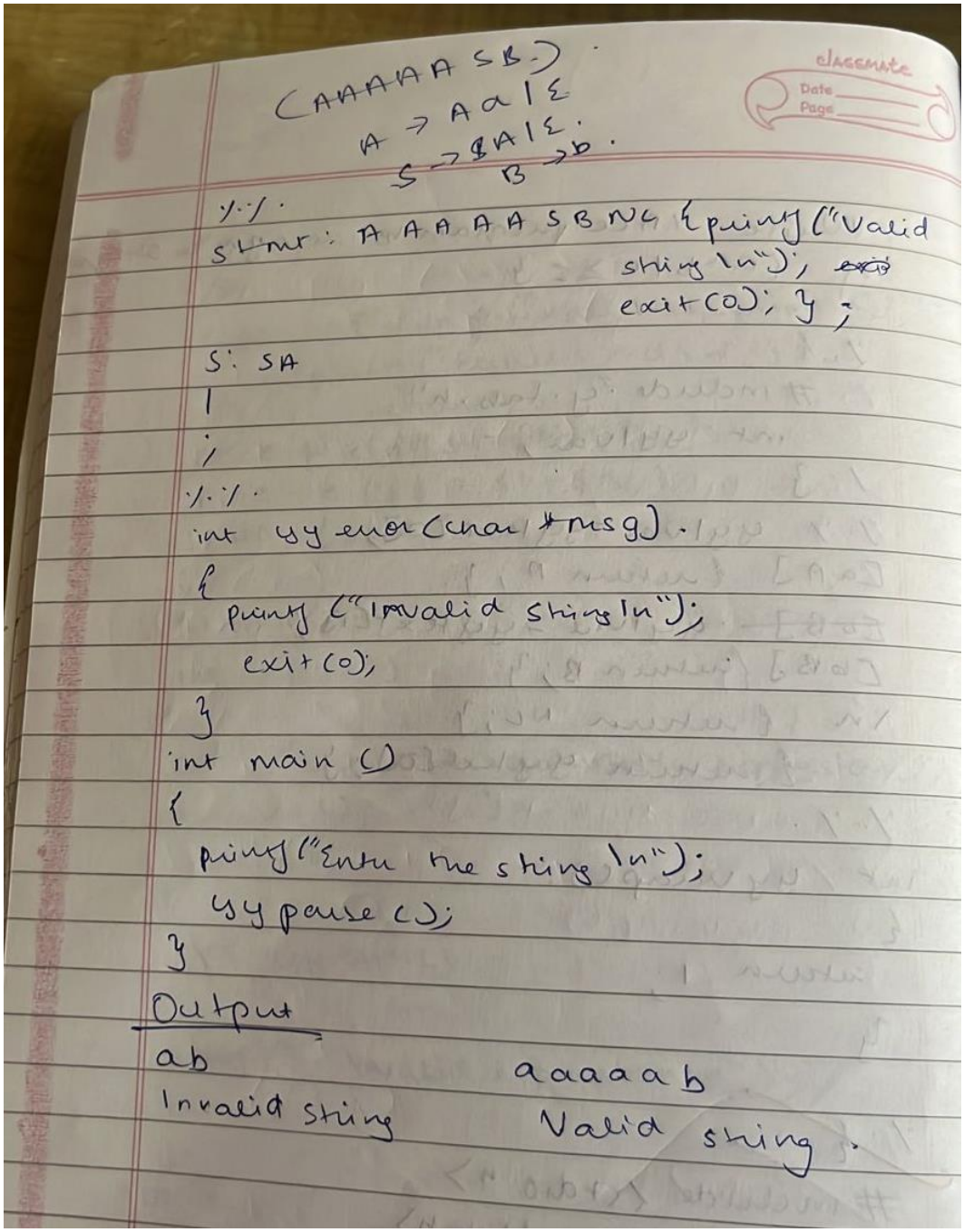
Code



Write YACC program to recognize strings of {a^n b | n ≥ 5}.

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

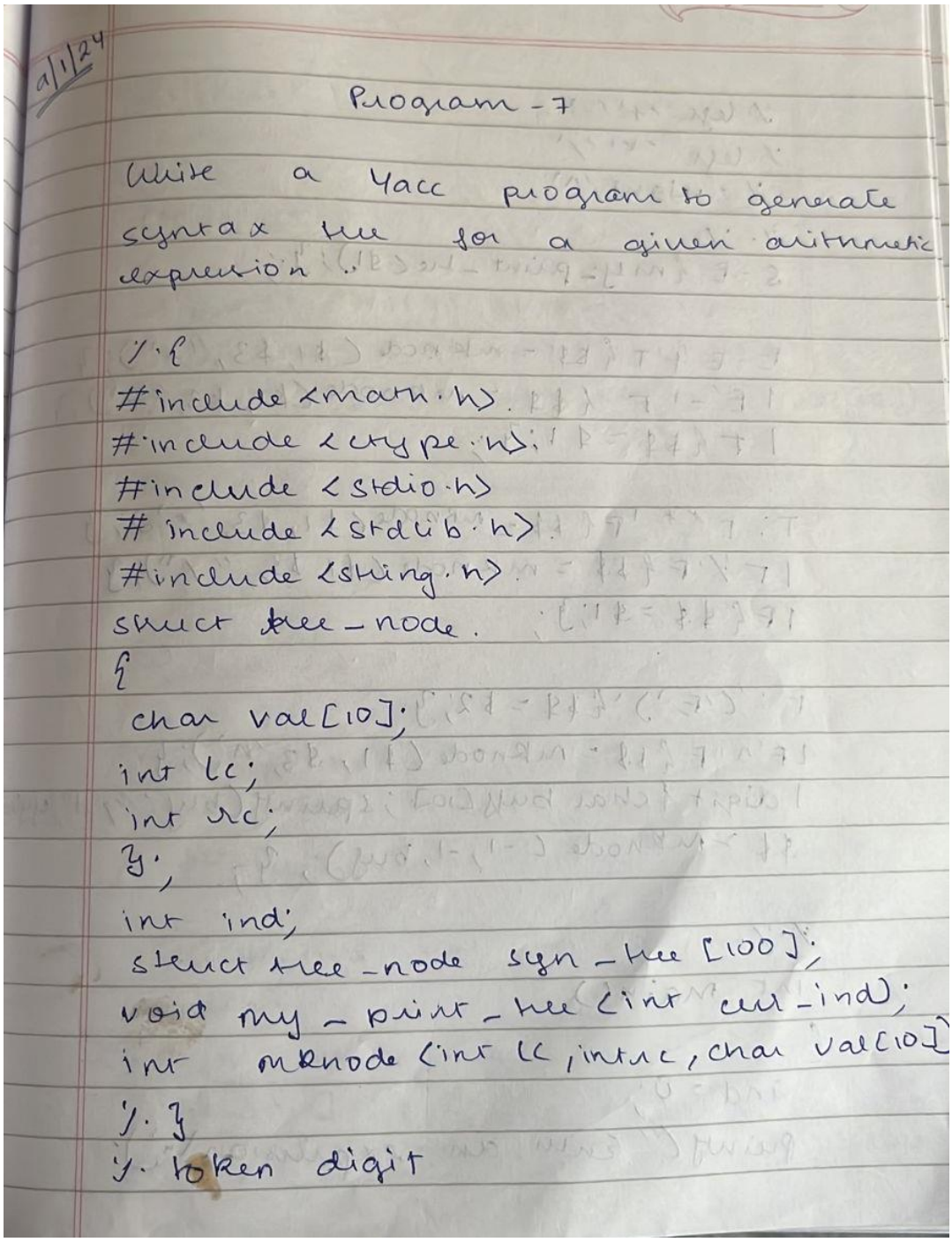
```
%{
#include <stdio.h>
#include <stdlib.h>
%}
%token A B NL
```



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.

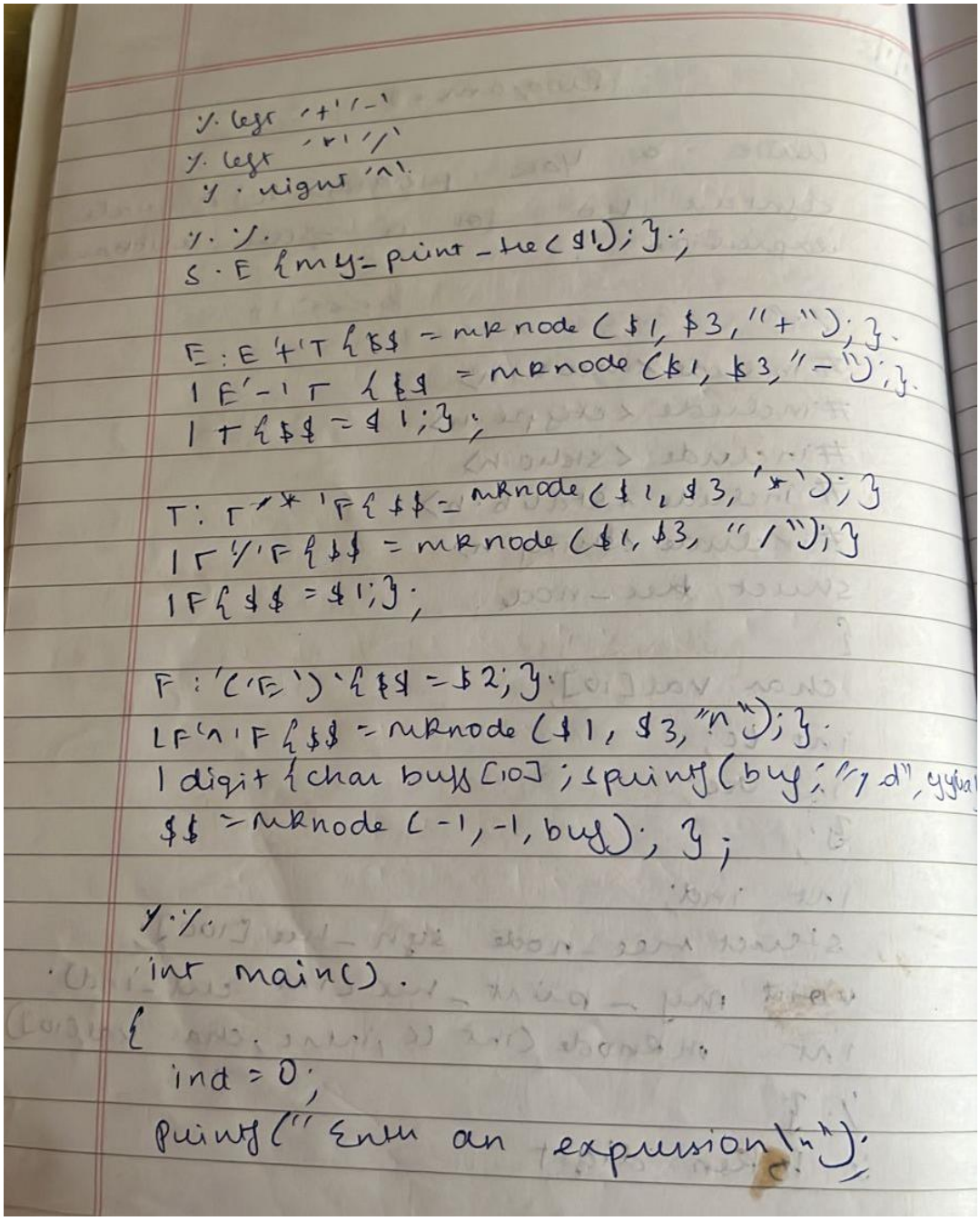


9/1/24

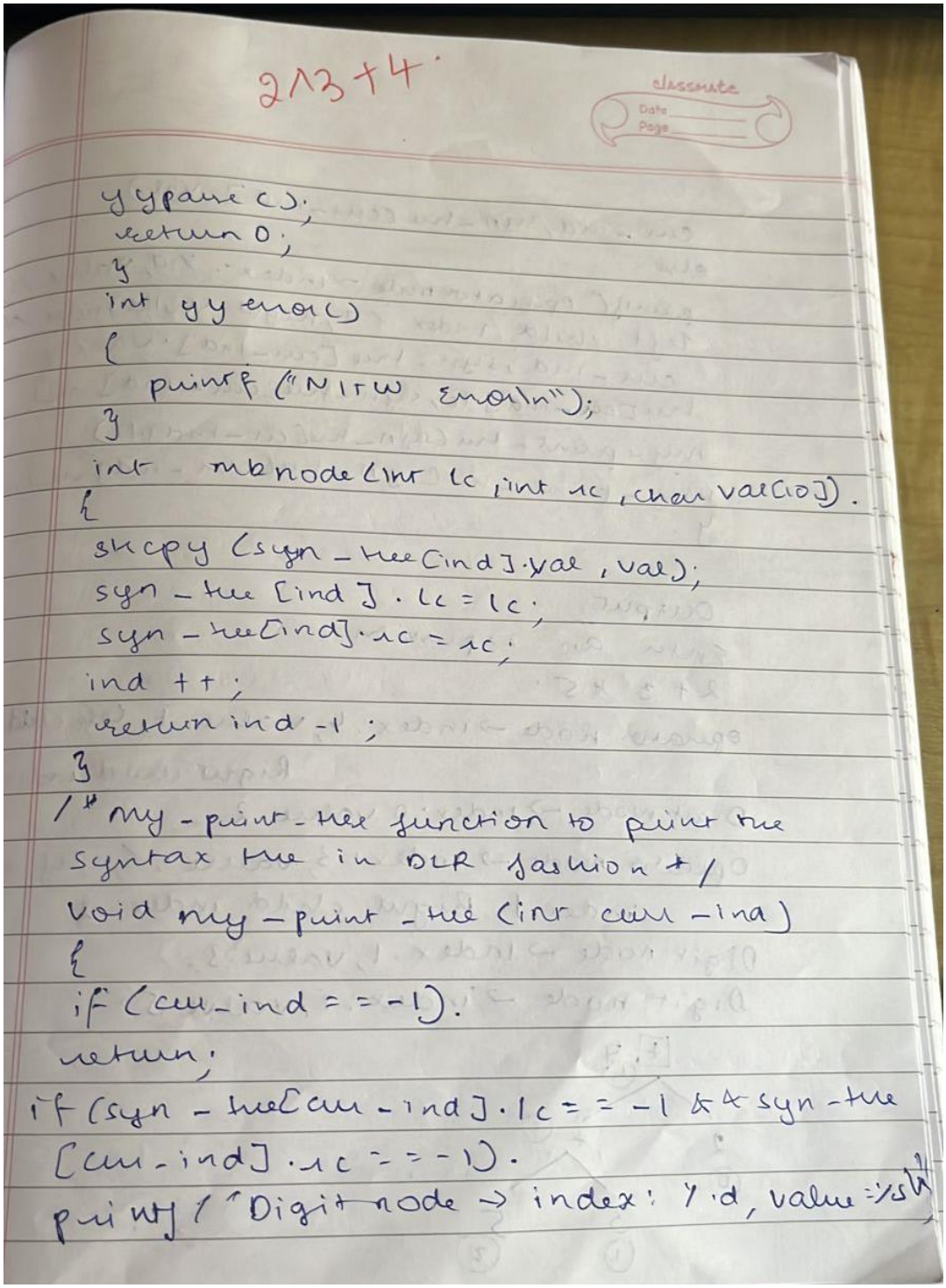
Program - 7

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

```
%{
#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
```

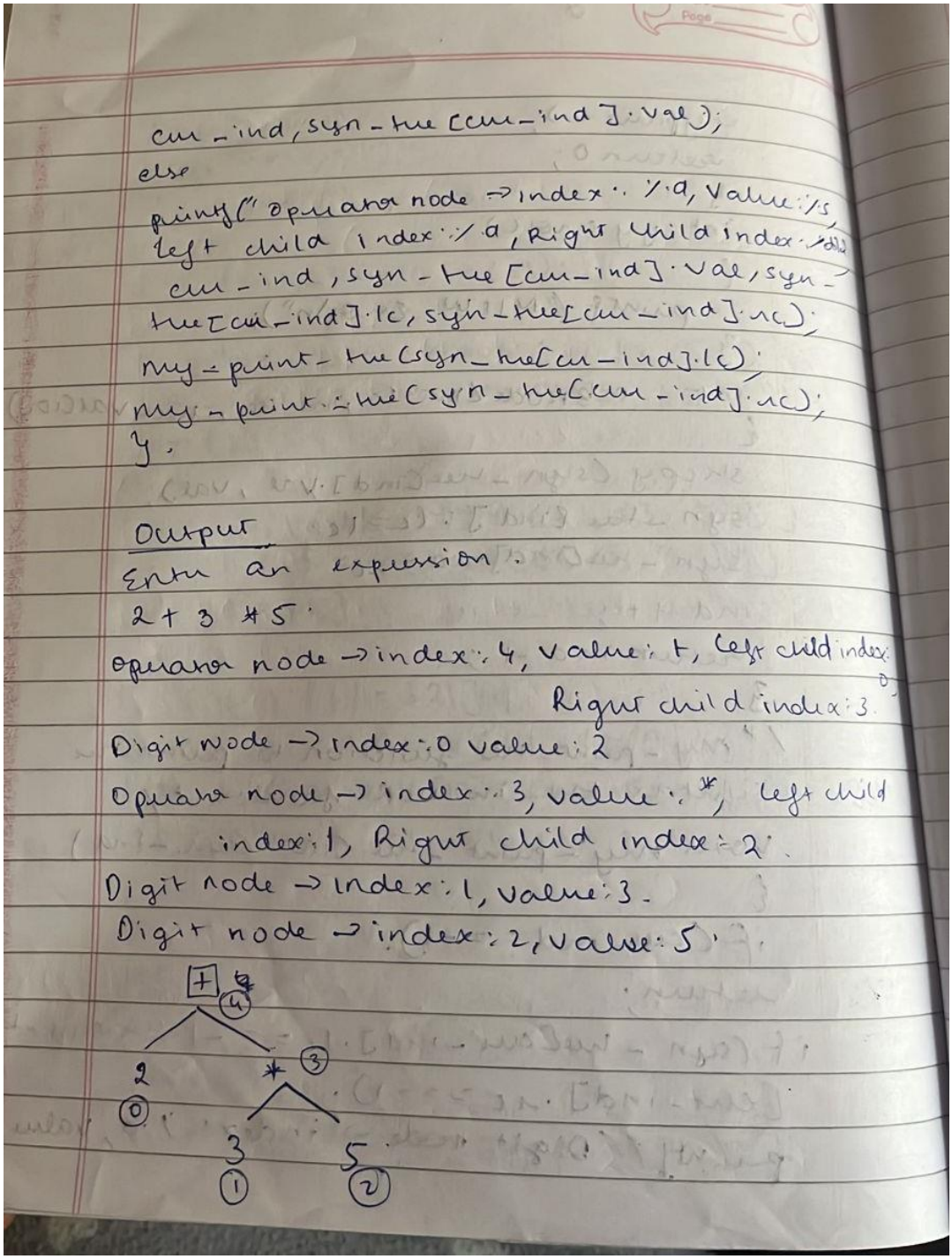


```
%left '+' '-'
%left '*' '/'
%right '^'

%%
S : E {my_print_tree($1); };

E : E '+' T { $$ = mknode($1, $3, "+"); }
| E '-' T { $$ = mknode($1, $3, "-"); }
| T { $$ = $1; };

T : T '*' F { $$ = mknode($1, $3, '*'); }
| T '/' F { $$ = mknode($1, $3, "/"); }
| F { $$ = $1; };

F : '(' E ')' { $$ = $2; }
| F '^' F { $$ = mknode($1, $3, "^"); }
| digit { char buff[10]; sprintf(buff, "%d", yyval
$$ = mknode(-1, -1, buff); };

%%
int main()
{
  ind = 0;
  printf("Enter an expression:\n");
```



2\3+4.

```
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;
}
/* my_print_tree function to print the
syntax tree in DLR fashion */
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",
```

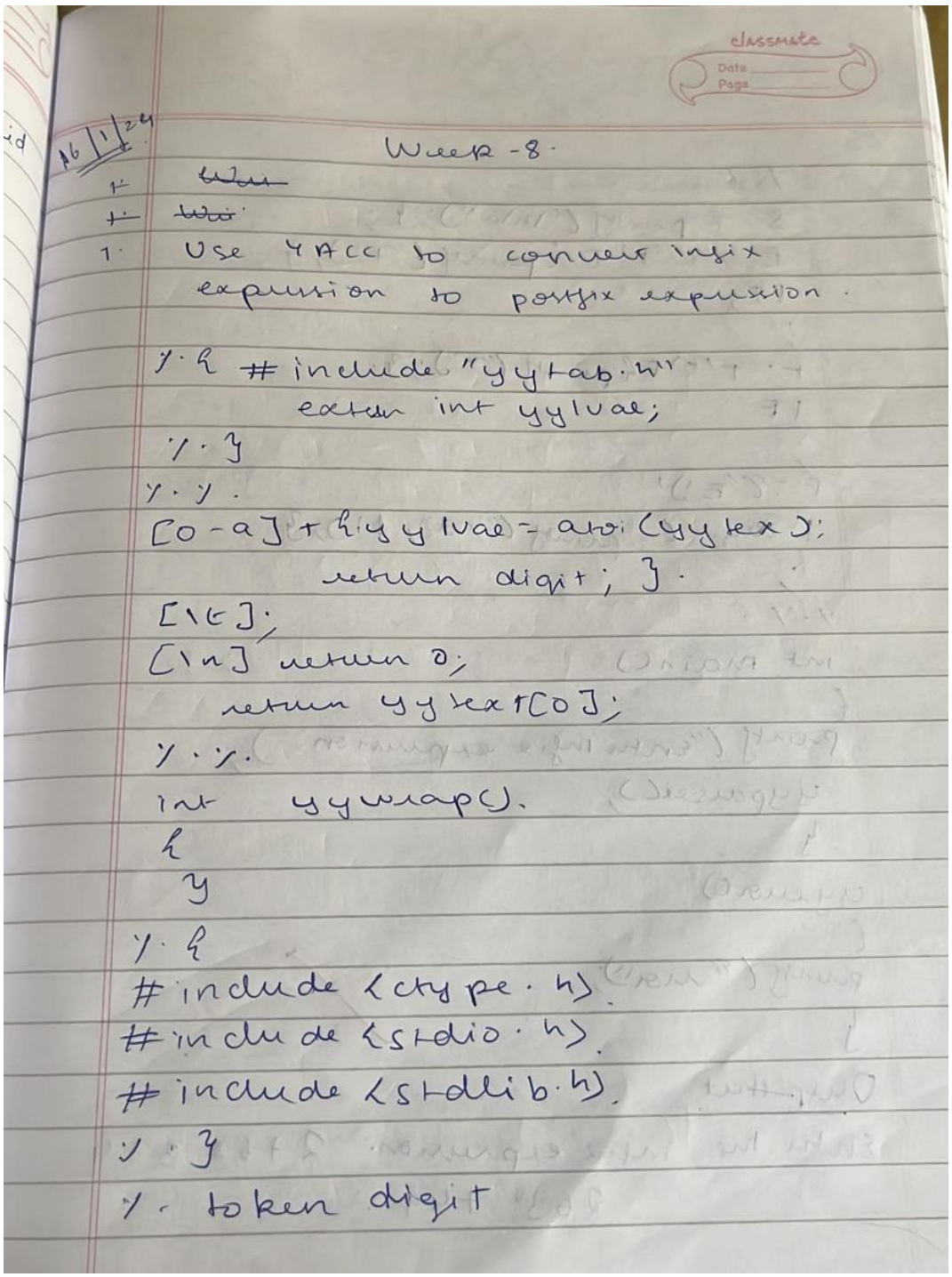


Output

```
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

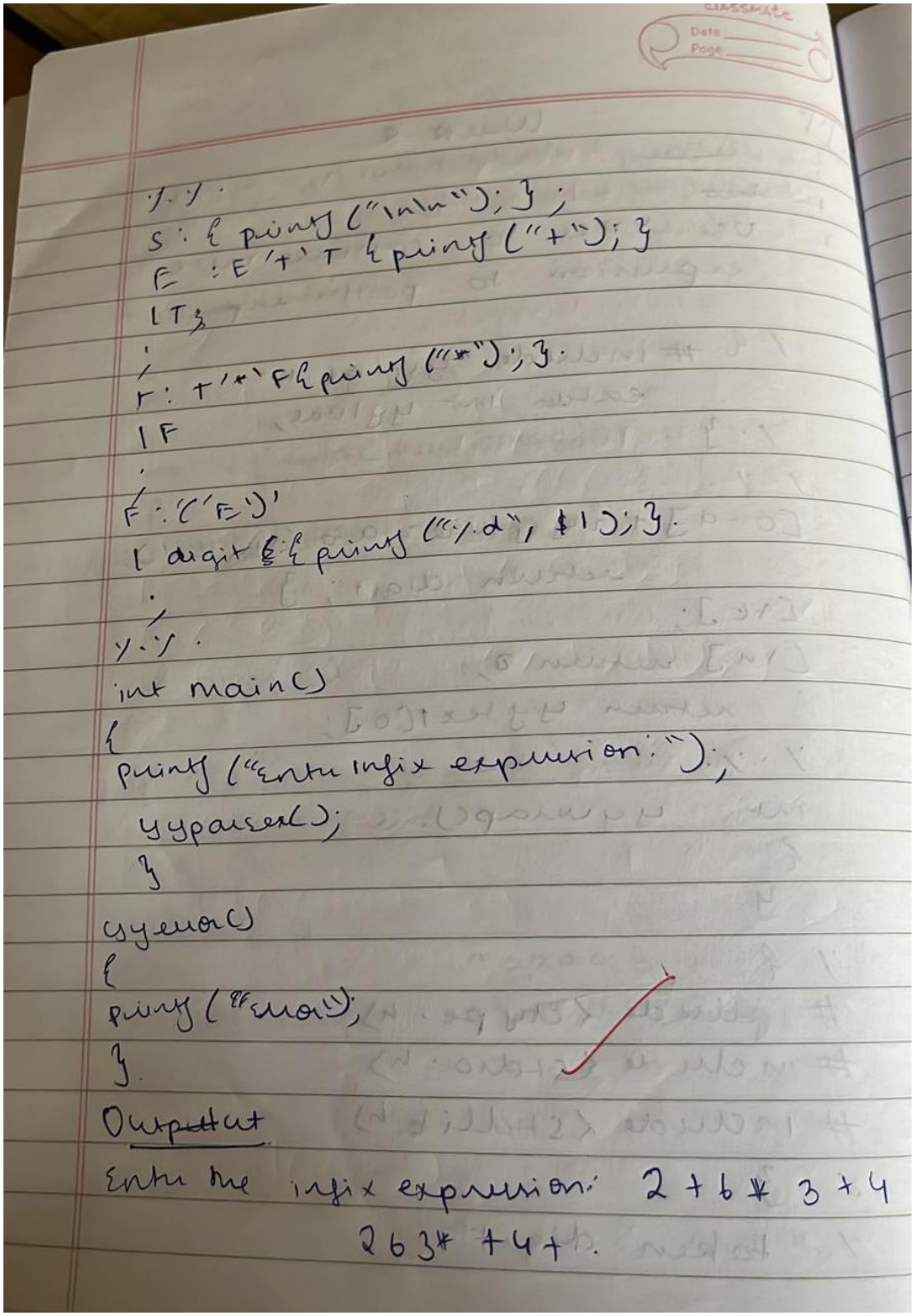
8.1 Write a program in YACC to convert infix to postfix expression.



Week -8.

1. Use YACC to convert infix expression to postfix expression.

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

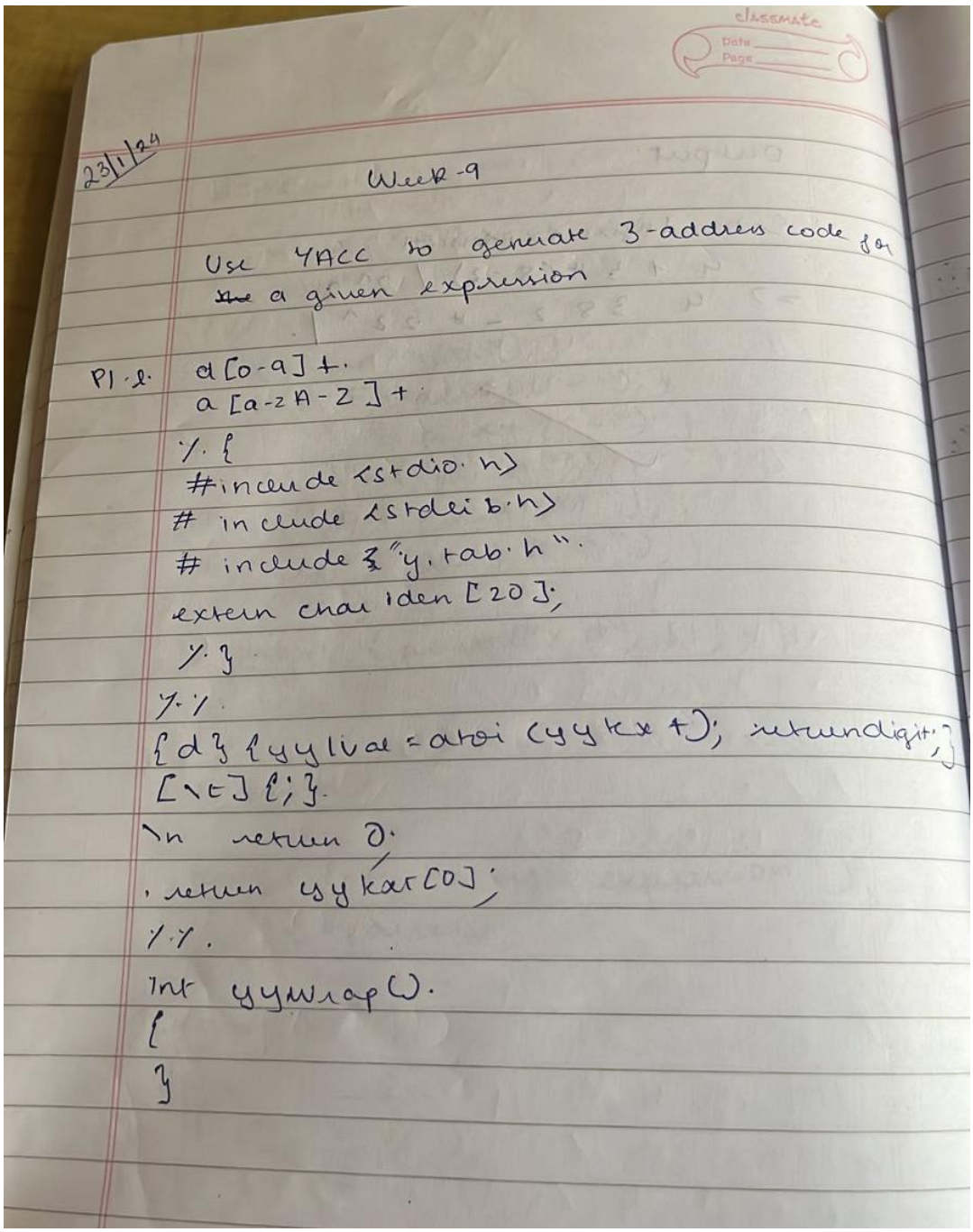


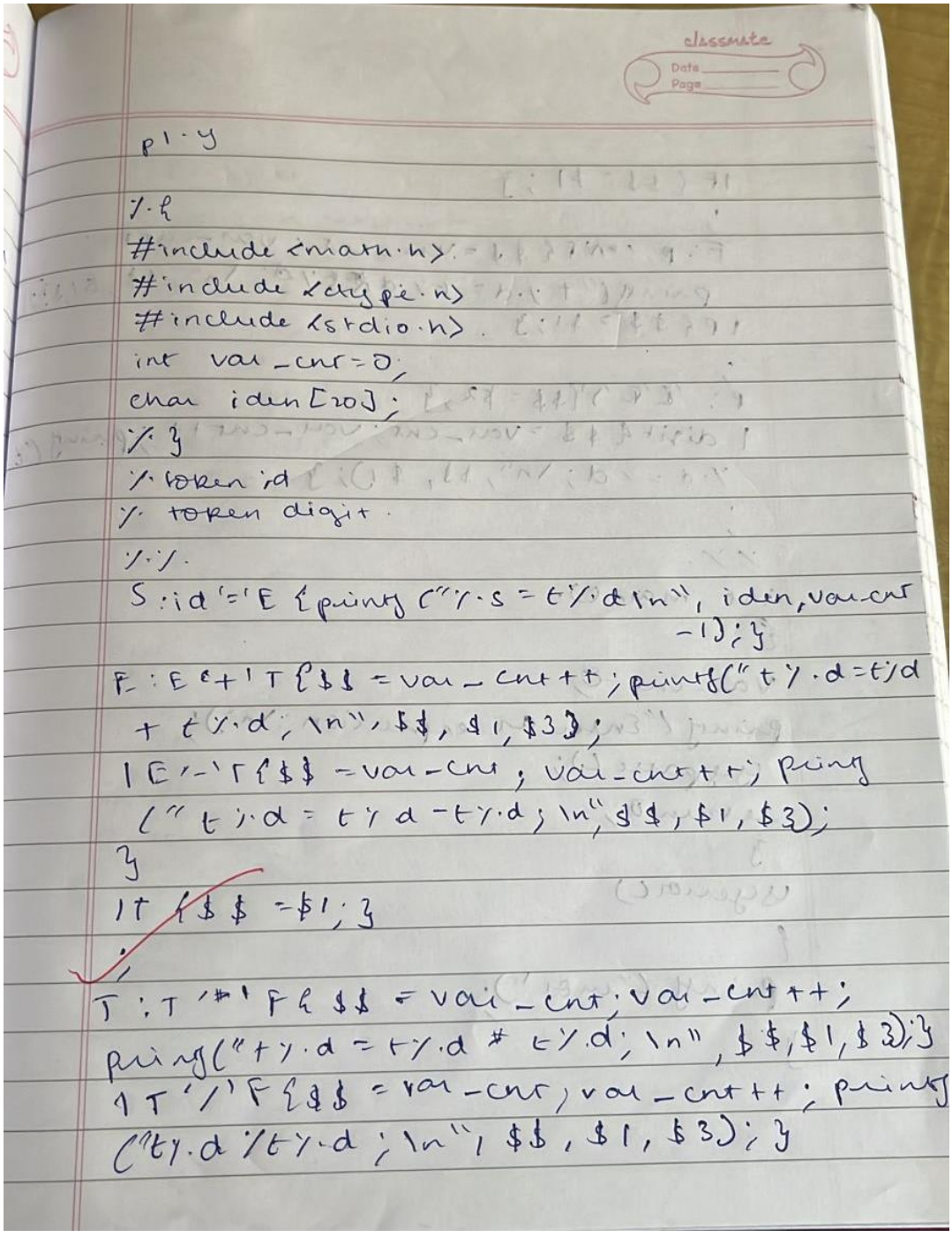
Output

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

Lab 9

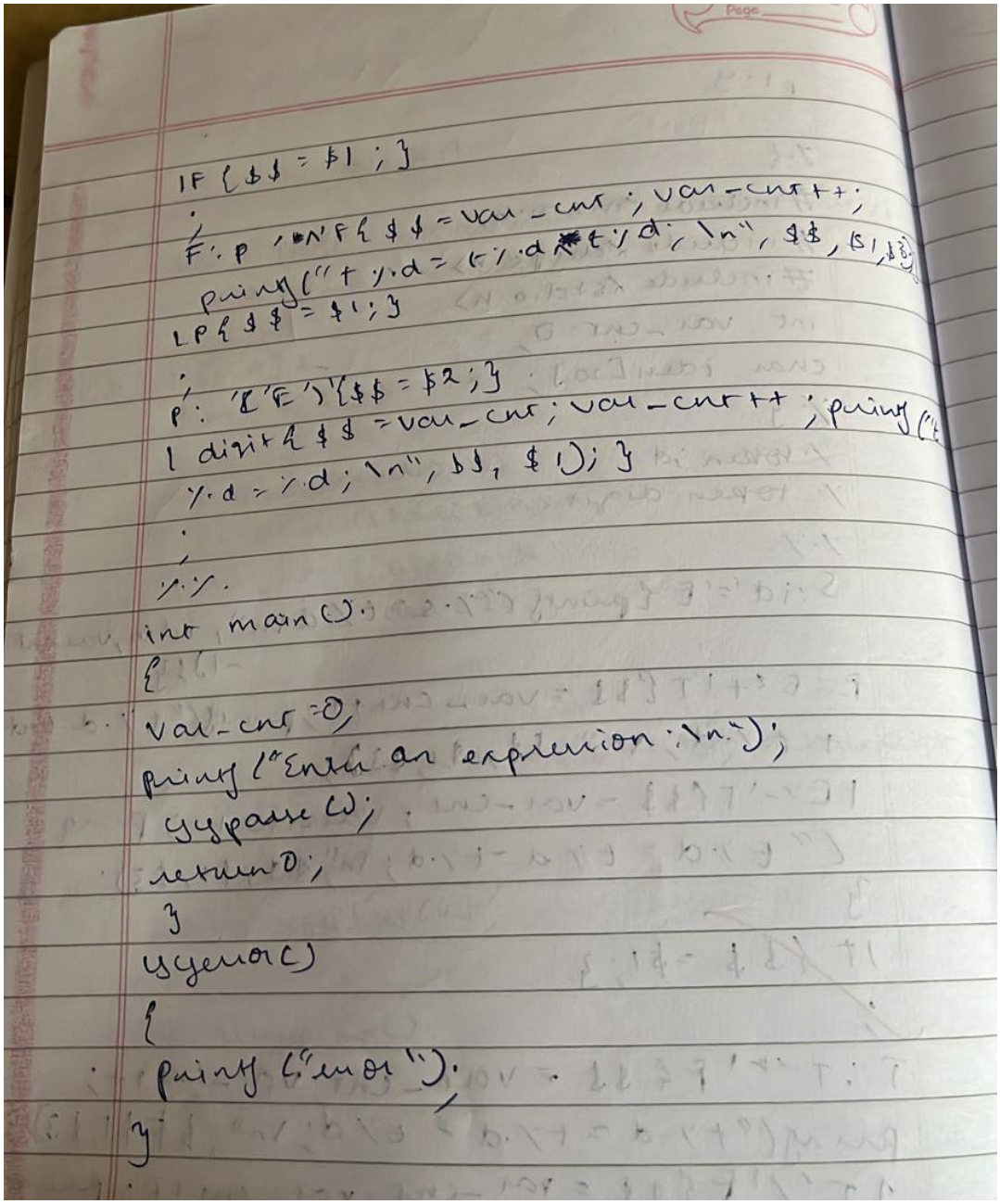
9.1 Write a program in YACC to generate three address code for a given expression.



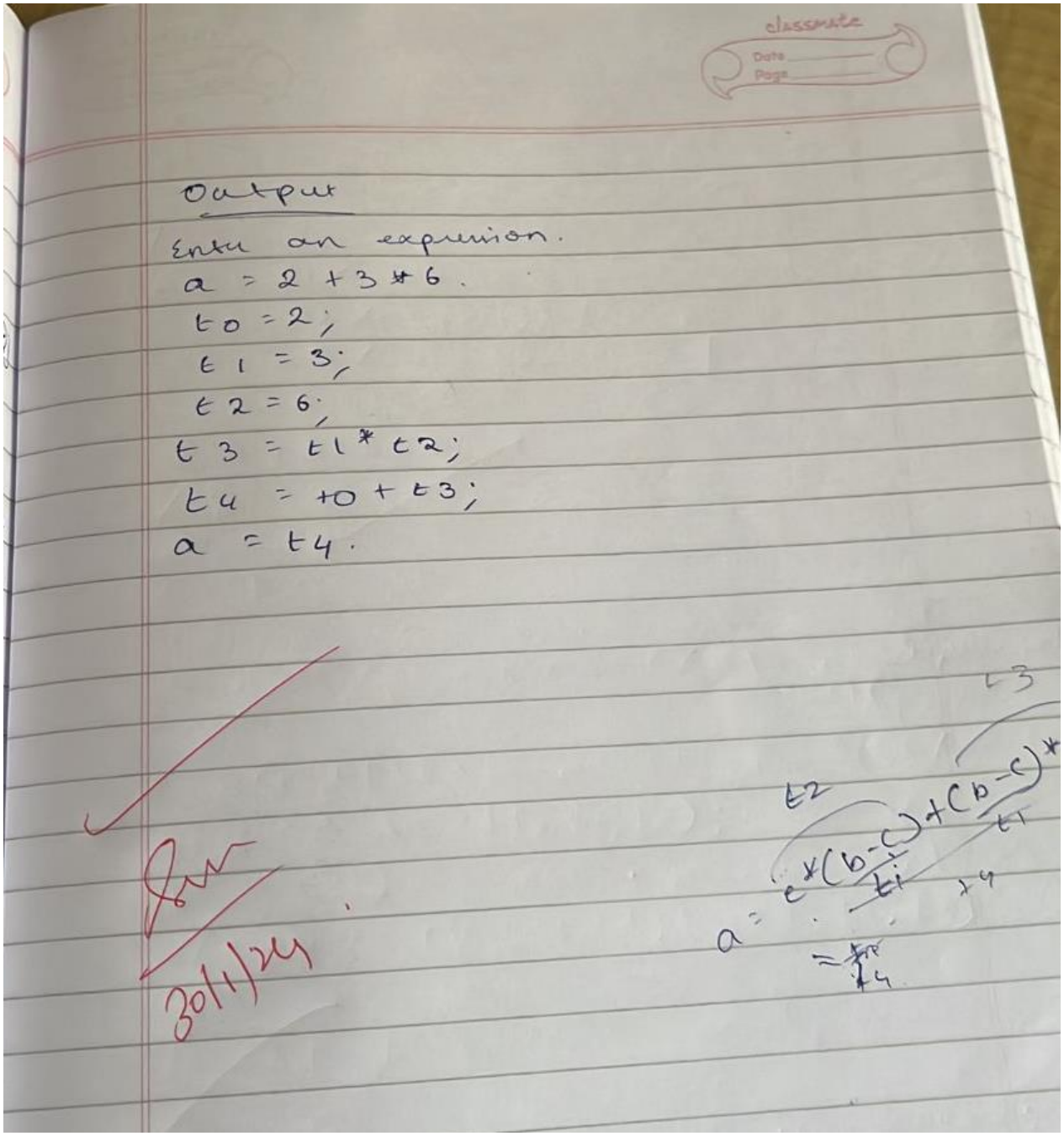


p1.y

%{
#include <math.h>
#include <ctype.h>
#include <stdio.h>
int var_cnt=0;
char iden[20];
%}
%token id
%token digit
%%
S:id'='E {printf("%s = t%d\n", iden, var_cnt-1);}
E: E'+'T {$$ = var_cnt++; printf("t%d = t%d + t%d; \n", $$, $1, $3);}
| E'-'T {$$ = var_cnt; var_cnt++; printf("t%d = t%d - t%d; \n", $$, $1, $3);}
}
| T {$$ = $1;}
;
T: T'*'F {$$ = var_cnt; var_cnt++;
printf("t%d = t%d * t%d; \n", $$, $1, $3);}
| T'/'F {$$ = var_cnt; var_cnt++; printf("t%d = t%d / t%d; \n", $$, $1, $3);}



```
IF {$$ = $1 ; }
;
F: P '+' F { $$ = var_cnt ; var_cnt++;
 printf("%d = %d * %d\n", $$, $1, $3);
LP { $$ = $1; }
;
P: '(' F ')' {$$ = $2;}
| digit { $$ = var_cnt; var_cnt++ ; printf("
 %d = %d\n", $$, $1); }
;
%%
int main()
{
var_cnt = 0;
printf("Enter an expression:\n");
yyparse();
return 0;
}
yyerror()
{
printf("error");
}
```



Output

Enter an expression.

a = 2 + 3 * 6.

t0 = 2;

t1 = 3;

t2 = 6;

t3 = t1 * t2;

t4 = t0 + t3;

a = t4.

30/1/24

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