VISVESVARAYA TECHNOLOGICAL UNIVERSITY

"JnanaSangama", Belgaum -590014, Karnataka.

LAB REPORT

on

COMPILER DESIGN

Submitted by

KANJIKA SINGH(1BM21CS086)

Under the Guidance of

Prof. Sunayana S.
Assistant Professor,
BMSCE

in partial fulfilment for the award of the degree of

BACHELOR OF ENGINEERING

in

COMPUTER SCIENCE AND ENGINEERING

B.M.S. COLLEGE OF ENGINEERING

(Autonomous Institution under VTU)

BENGALURU-560019

November 2023-February 2024

B. M. S. College of Engineering,

Bull Temple Road, Bangalore 560019

(Affiliated To Visvesvaraya Technological University, Belgaum)

Department of Computer Science and Engineering

CERTIFICATE

This is to certify that the Lab work entitled "**Compiler Design**" carried out by **Kanjika Singh (1BM21CS086)** , who is bonafide student of **B. M. S. College of Engineering.** It is in partial fulfilment for the award of **Bachelor of Engineering in Computer Science and Engineering** of the Visvesvaraya Technological University, Belgaum during the year 2023-24.

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

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

B. M. S. COLLEGE OF ENGINEERING

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

DECLARATION

I, Kanjika Singh (1BM21CS086), student of 5th Semester, B.E, Department of Computer Science and Engineering, B. M. S. College of Engineering, Bangalore, here by declare that, this lab report entitled " **Compiler Design**" has been carried out by me under theguidance of Prof. Sunayana S., Assistant Professor, Department of CSE, B. M. S. College ofEngineering, Bangalore during the academic semester November-2023-February-2024.

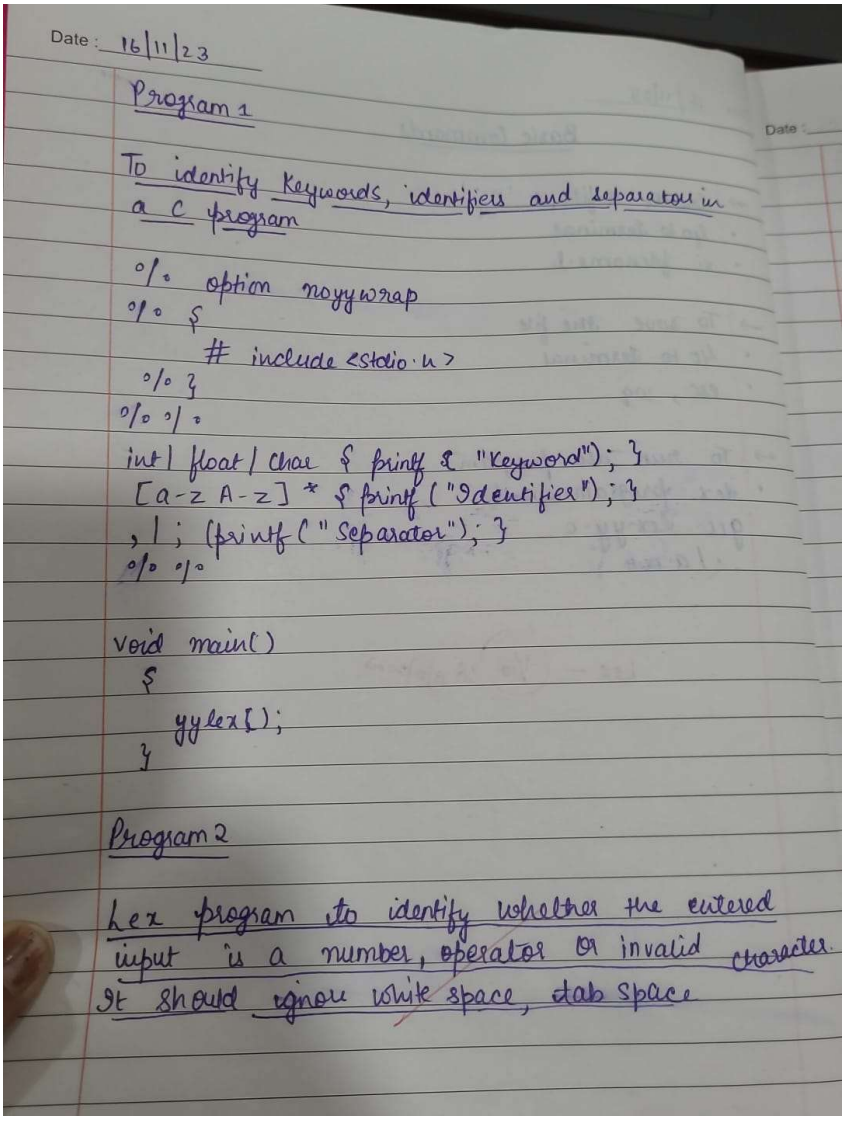
I also declare that to the best of my knowledge and belief, the development reported here is not from part of any other report by any other students.

TABLE OF CONTENTS

Lab 1

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

Code:



Date : 16/11/23

Program 1

To identify keywords, identifiers and separators in a C program

```
%{ option noyywrap
%{ $
    # include <stdio.h>
 %} 
%% %%
int | float | char { printf ("Keyword"); }
[a-z A-z] * { printf ("Identifier"); }
, | ; {printf ("Separator"); }
%% %%

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

Program 2

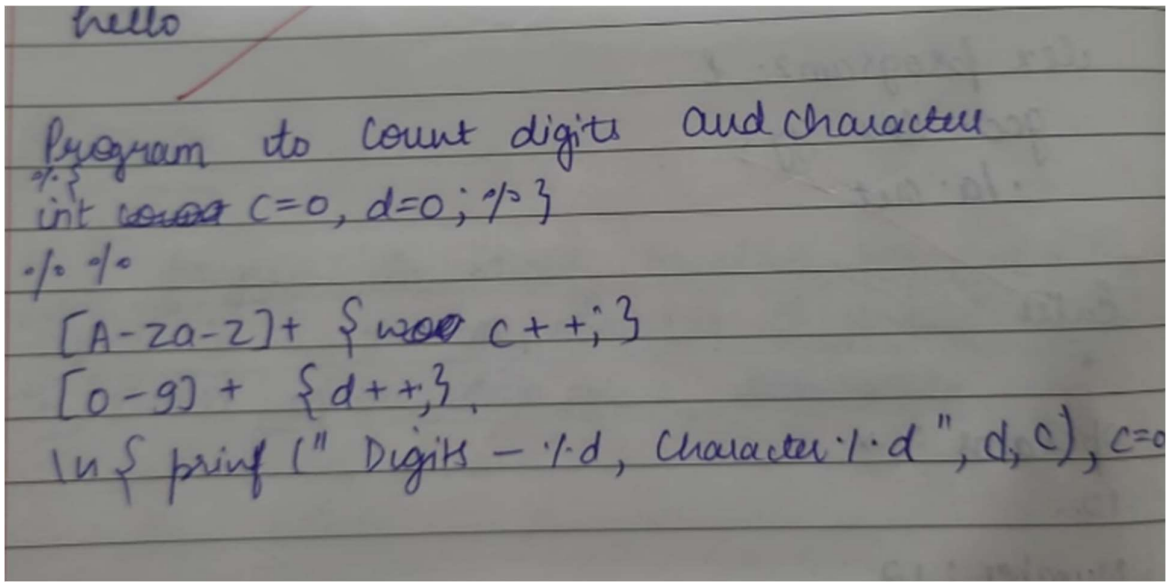
Lex program to identify whether the entered input is a number, operator or invalid character. It should ignore white space, tab space

Output

```
kanjika@ubuntu:~/Desktop/lab practice$ lex lab1a.l
kanjika@ubuntu:~/Desktop/lab practice$ gcc lex.yy.c
kanjika@ubuntu:~/Desktop/lab practice$ ./a.out
Enter the text int main a 9;
 int - Identifier
 main - Keyword
 a - Identifier
 9 - Digit
;- Separator
```

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

Code

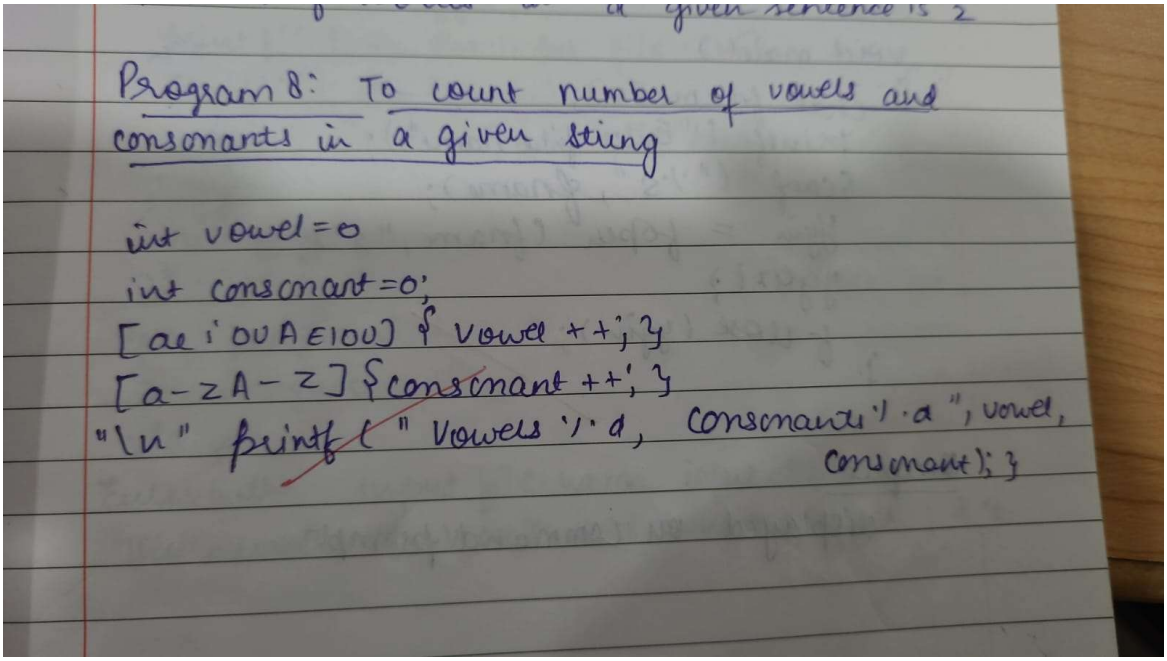


Output

```
kanjika@ubuntu:~/Desktop/lab practice$ lex week1_countDigitsandCharacters.l
kanjika@ubuntu:~/Desktop/lab practice$ gcc lex.yy.c
kanjika@ubuntu:~/Desktop/lab practice$ ./a.out
Enter input
hey, my address is 6 main road, 221005
The number of digits in the sentence is 7 and characters is 22
```

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

Code



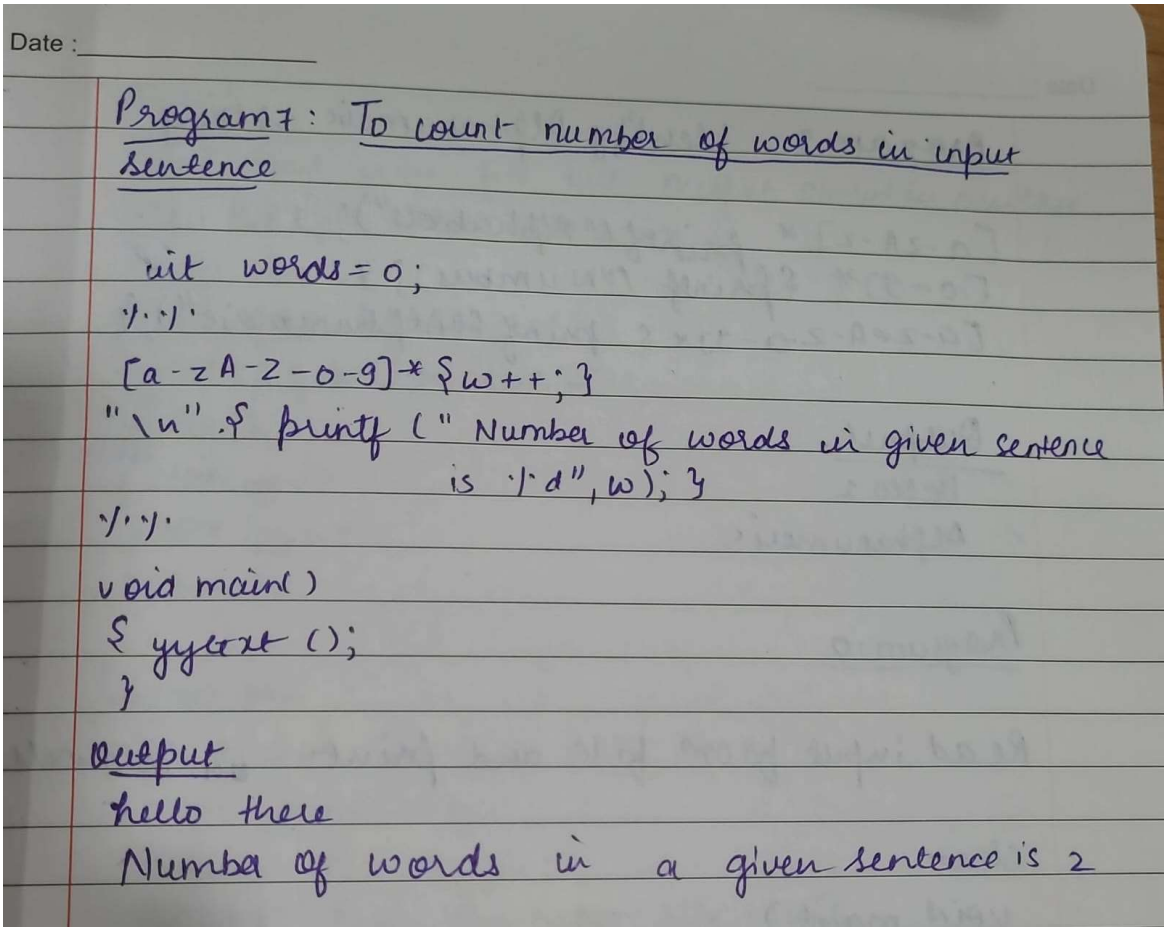
Output

```
kanjika@ubuntu:~/Desktop/lab practice$ lex week1_NoofVowelsConsonants.l
kanjika@ubuntu:~/Desktop/lab practice$ gcc lex.yy.c
kanjika@ubuntu:~/Desktop/lab practice$ ./a.out
Enter input
hello my name is kanjika
The number of vowels in the sentence is 8 and consonants is 12
```

Lab 2

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

Code



Date :

Program 7 : To count number of words in input sentence

```
int words = 0;
%%
[a-zA-Z-0-9]+ {w++;}
"\n" {printf("Number of words in given sentence
                is %d", w);}
%%
void main()
{ yylex();
}
```

Output

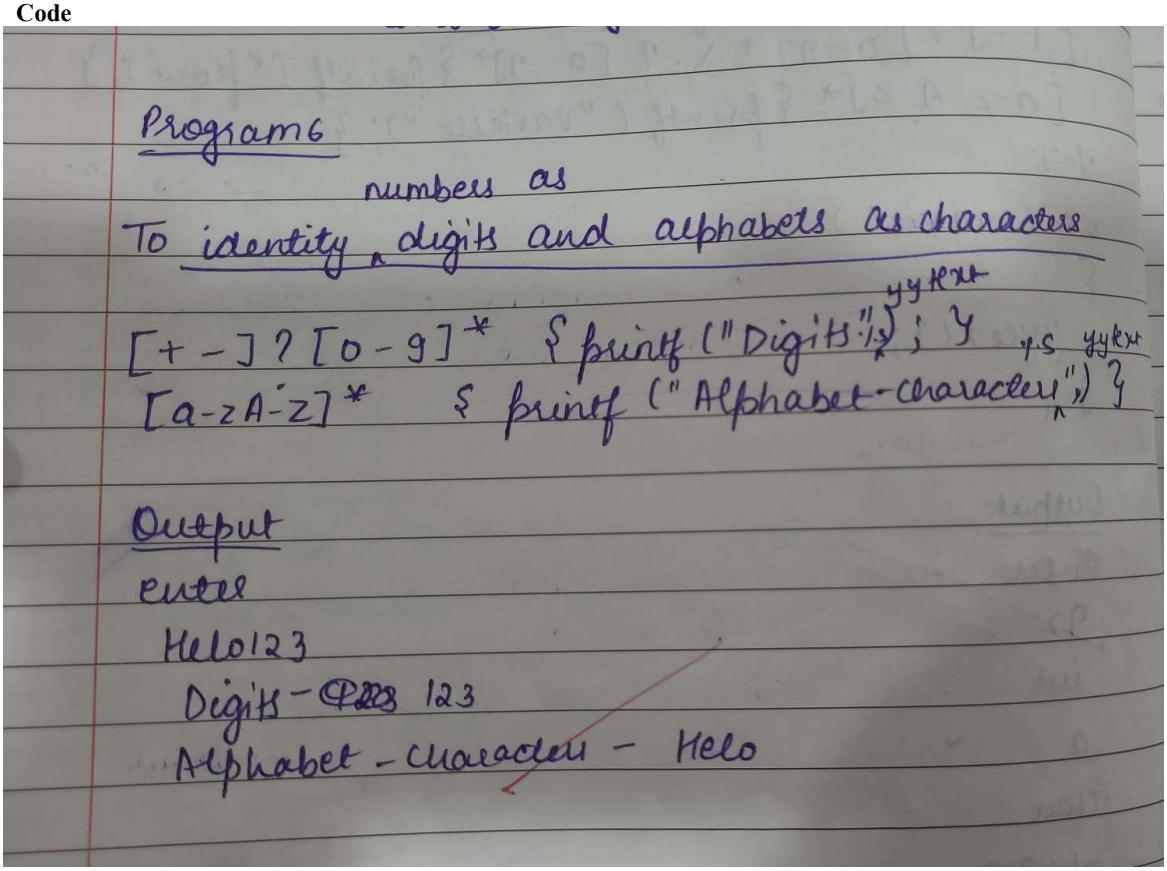
```
hello there
Numba of words in a given sentence is 2
```

Output

```
kanjika@ubuntu:~/Desktop/lab practice$ lex week2_noOfwords.l
kanjika@ubuntu:~/Desktop/lab practice$ gcc lex.yy.c
kanjika@ubuntu:~/Desktop/lab practice$ ./a.out
Enter input
hey this is test input
    The number of words in sentence is 5
this is life
  The number of words in sentence is 3
```

2.2 Write a program in lex to demonstrate regular definition.

Code

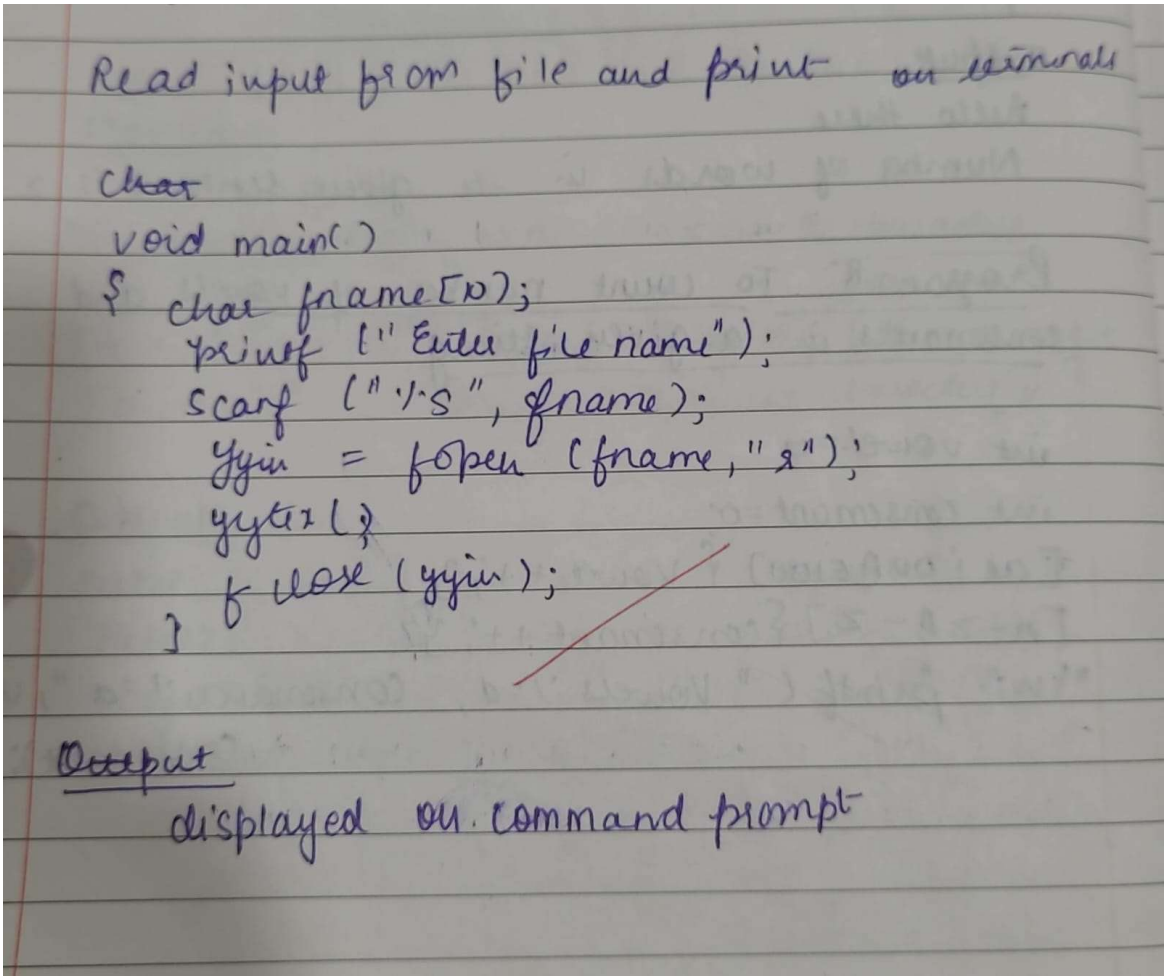


Output

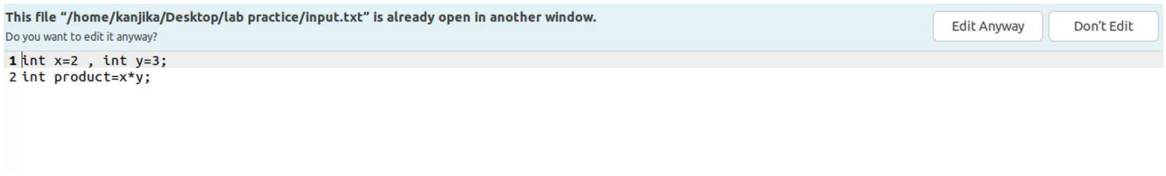
```
kanjika@ubuntu:~/Desktop/lab practice$ lex week2_identifyCharDigits.l
kanjika@ubuntu:~/Desktop/lab practice$ gcc lex.yy.c
kanjika@ubuntu:~/Desktop/lab practice$ ./a.out
Enter input
hello
characters

78
Digits

gwhdh982
Invalid input
```

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

Code



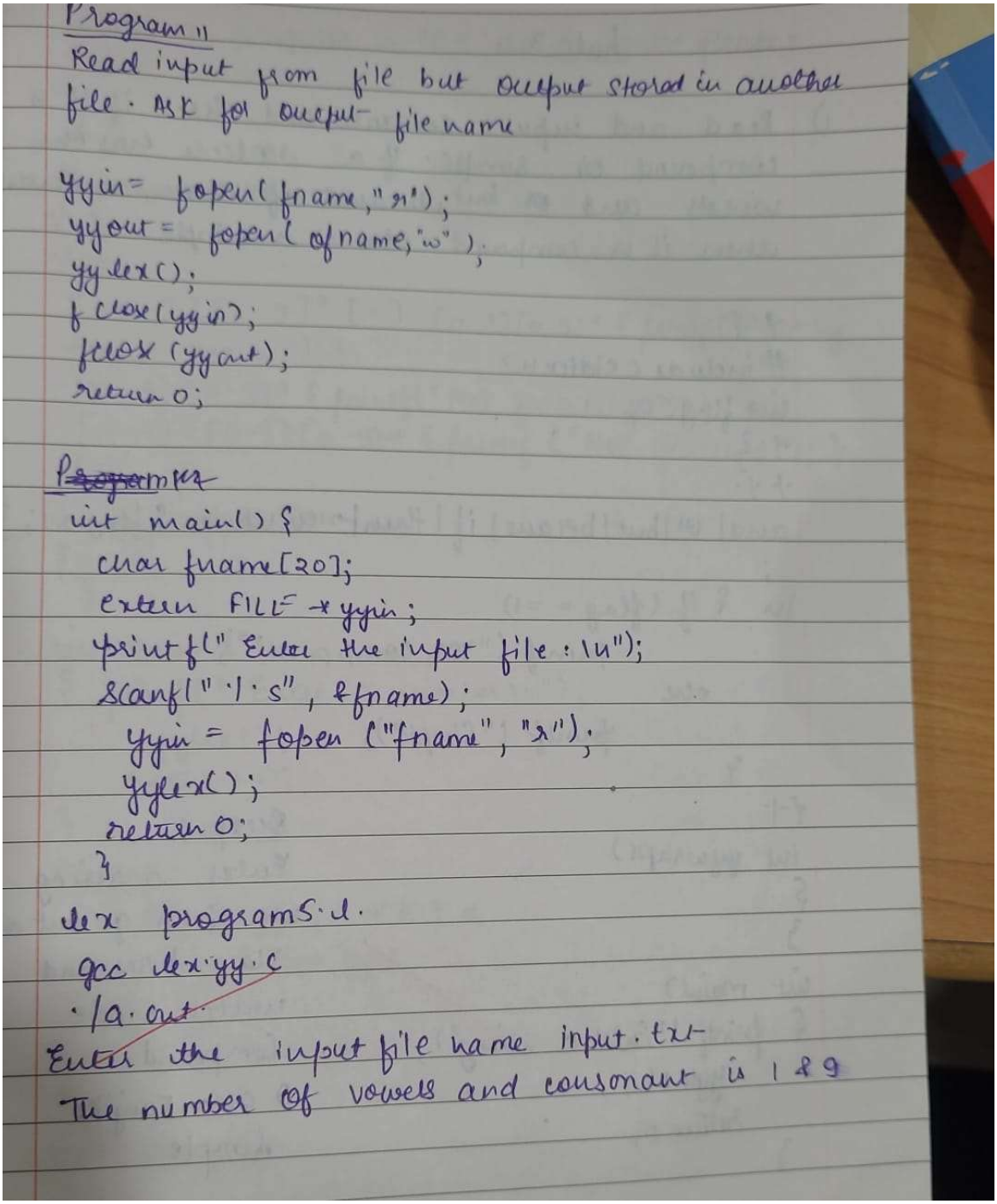
Output




```
kanjika@ubuntu:~/Desktop/lab practice$ lex week2_Fileinput.l
kanjika@ubuntu:~/Desktop/lab practice$ gcc lex.yy.c
kanjika@ubuntu:~/Desktop/lab practice$ ./a.out
int is a keyword.
 x is an identifier.
= is an assignment operator.
2 is/are digit(s).
 , is a separator.
 int is a keyword.
 y is an identifier.
= is an assignment operator.
3 is/are digit(s).
; is a delimiter.
int is a keyword.
 product 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



Program 11

Read input from file but output stored in another file. Ask for output file name

```
yyin= fopen(fname,"r");
yyout = fopen(ofname,"w");
yylex();
fclose(yyin);
fclose(yyout);
return 0;
```

Program

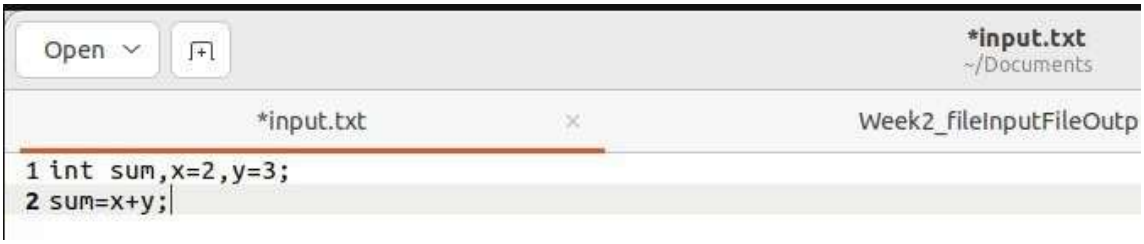
```
int main() {
  char fname[20];
  extern FILE *yyin;
  printf("Enter the input file : \n");
  scanf("%s", &fname);
   yyin = fopen("fname", "r");
   yylex();
  return 0;
  }
```

```
lex programs.l
gcc lex.yy.c
./a.out
```

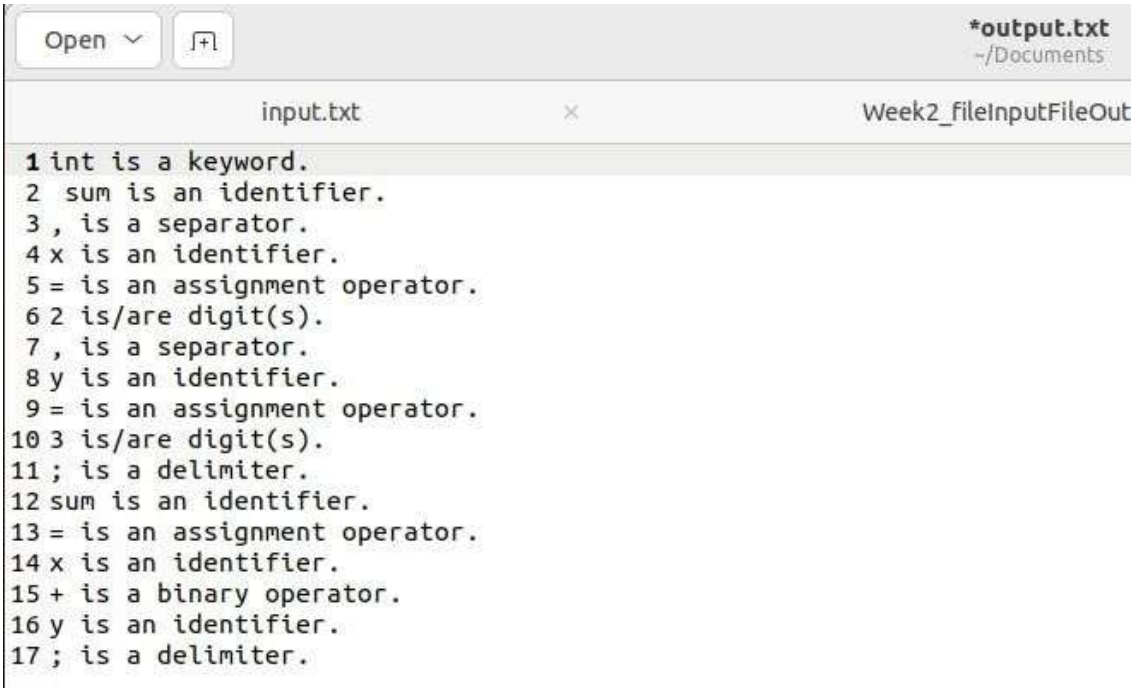
Enter the input file name input.txt

The number of vowels and consonant is 1 & 9

Output

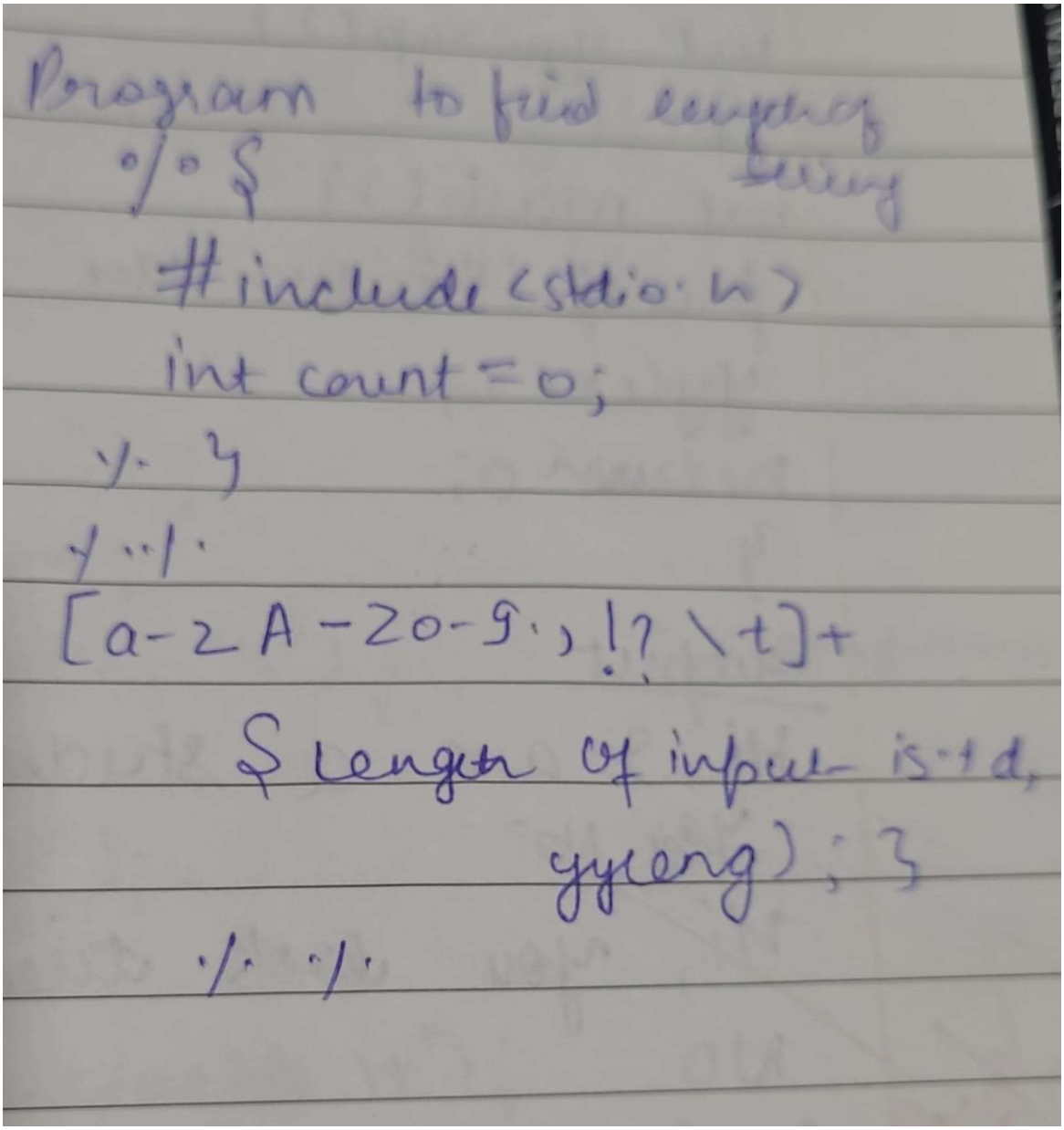



```
Printed in output.txt
```



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

Code



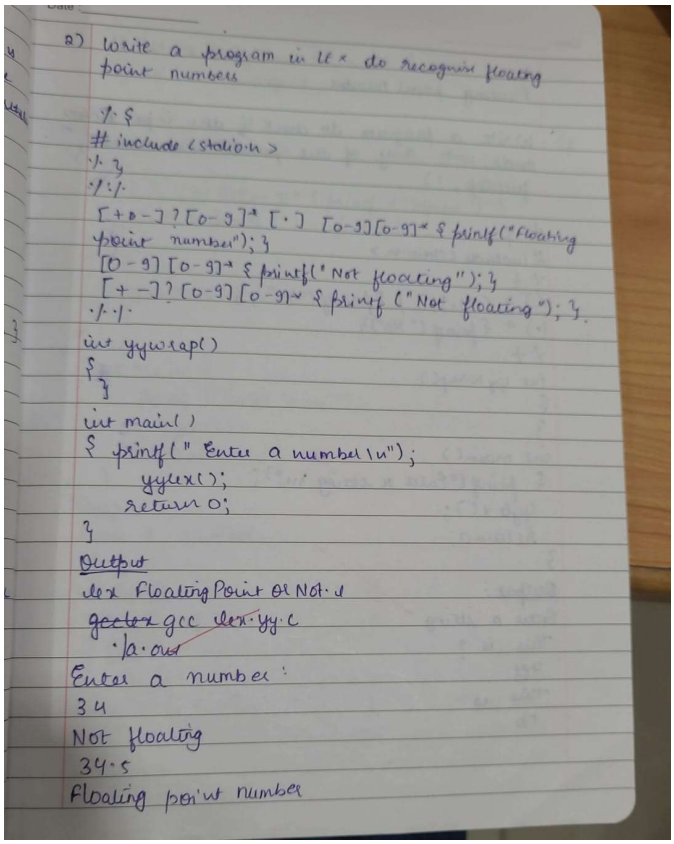
Output

```
neha29@neha-VirtualBox:~/Documents$ lex Week2_lengthofString.l
neha29@neha-VirtualBox:~/Documents$ gcc lex.yy.c
neha29@neha-VirtualBox:~/Documents$ ./a.out
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



```
2) Write a program in LEX to recognize floating
point numbers

%{
#include<stdio.h>
%}
%%
[+|-]?[0-9]*[.][0-9][0-9]* {printf("Floating
point number");}
[0-9][0-9]* {printf("Not floating");}
[+|-]?[0-9][0-9]* {printf("Not floating");}
%%
int yywrap()
{
}
int main()
{ printf("Enter a number\n");
    yylex();
    return 0;
}
Output
lex FloatingPoint or Not.l
gcc lex.yy.c
./a.out
Enter a number:
34
Not floating
34.5
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

```
%{
#include<stdio.h>
int flag=0;
%}
%%
and|or|but|because|if|then|nevertheless {flag=1;}
;
\n {if(flag==1)
        printf("yes, compound!");
    else
        printf("simple");
    }
%%
int yywrap()
{
}
int main()
{ printf("Enter a string");
    yylex();
    return 0;
}
```

Output

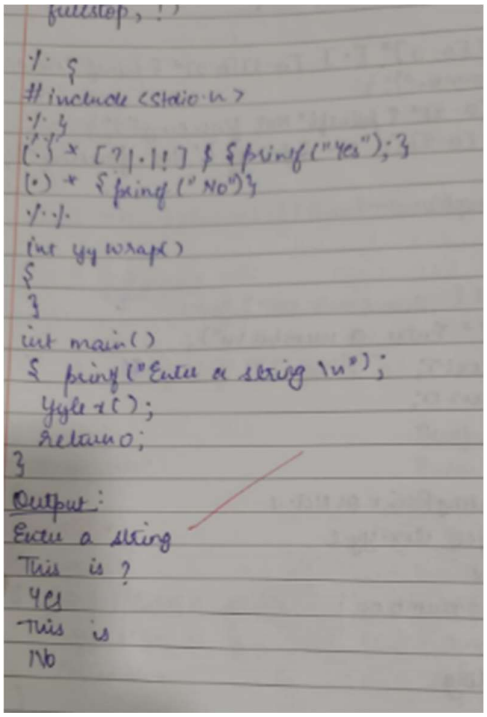
```
Enter a string
we either go out or
stay home and
watch movie.
Compound
I am Ranjika
simple
```

Output

```
neha29@neha-VirtualBox:~/Documents$ lex Week3_compoundOrSimple.l
neha29@neha-VirtualBox:~/Documents$ gcc lex.yy.c
neha29@neha-VirtualBox:~/Documents$ ./a.out
Enter a sentence:
This is a car.
Simple sentence!
neha29@neha-VirtualBox:~/Documents$ gcc lex.yy.c
neha29@neha-VirtualBox:~/Documents$ ./a.out
Enter a sentence:
She is good at singing and dancing.
Compound sentence!
neha29@neha-VirtualBox:~/Documents$
```

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

Code



```
fullstop, !)
%{
#include <stdio.h>
%}
%%
[.]*[?|.|!] { printf("Yes"); }
(.)* { printf("No"); }
%%
int yywrap()
{
}
int main()
{ printf("Enter a string \n");
  yylex();
  return 0;
}
```

Output:

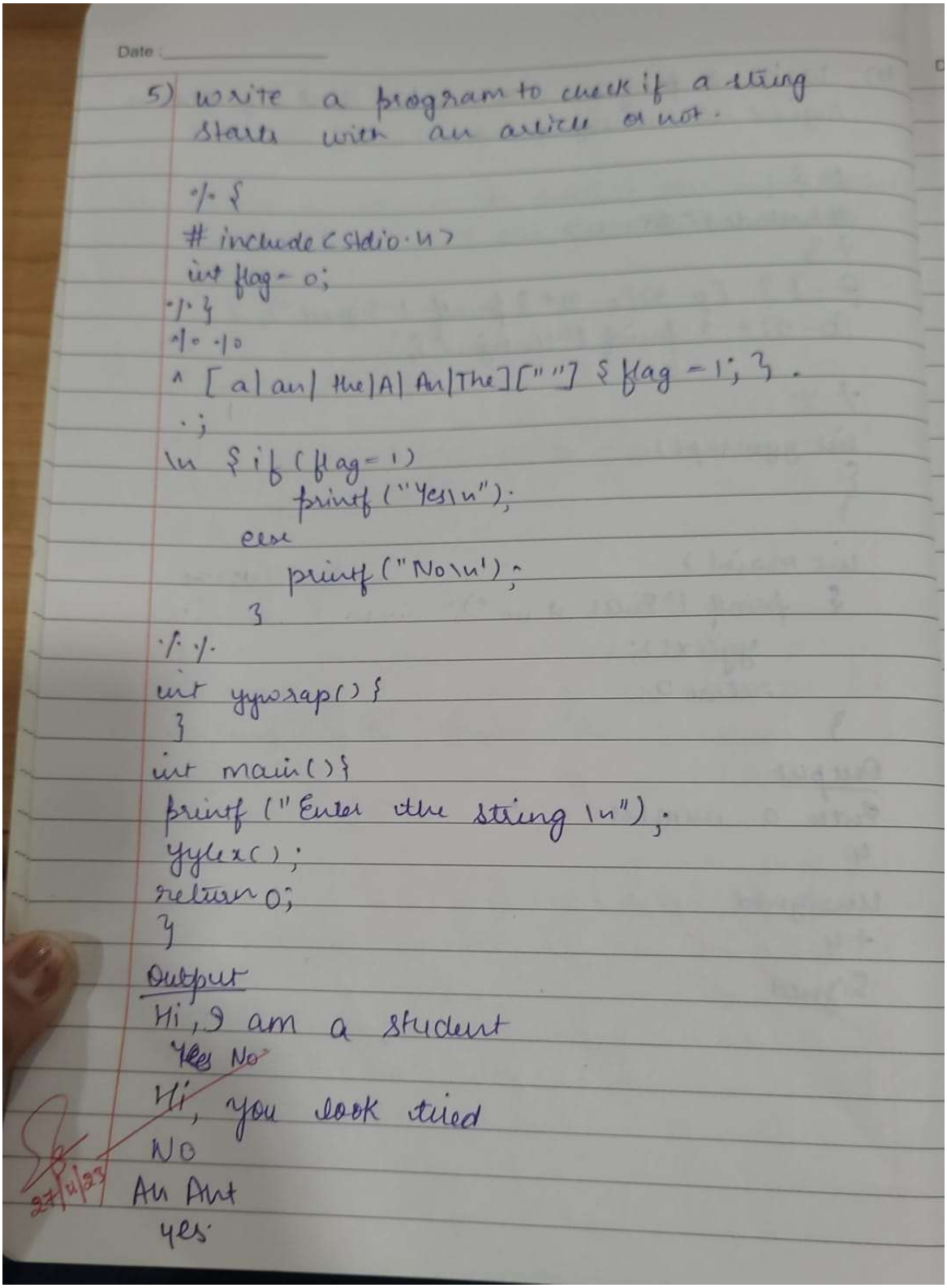
```
Enter a string
This is ?
Yes
This is
No
```

Output

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

5) write a program to check if a string starts with an article or not.

```
%{
#include<stdio.h>
int flag=0;
%}
%%
^[a|an|the|A|An|The][" "] {flag=1;}.
.;
\n {if(flag=1)
      printf("Yes\n");
   else
      printf("No\n");
   }
%%
int yywrap(){
}
int main(){
printf("Enter the string\n");
yylex();
return 0;
}
```

Output

Hi, I am a student

~~Yes~~ No

Hi, you look tired

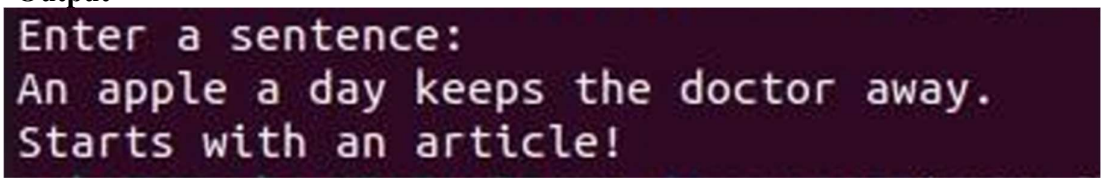
NO

An Ant

yes

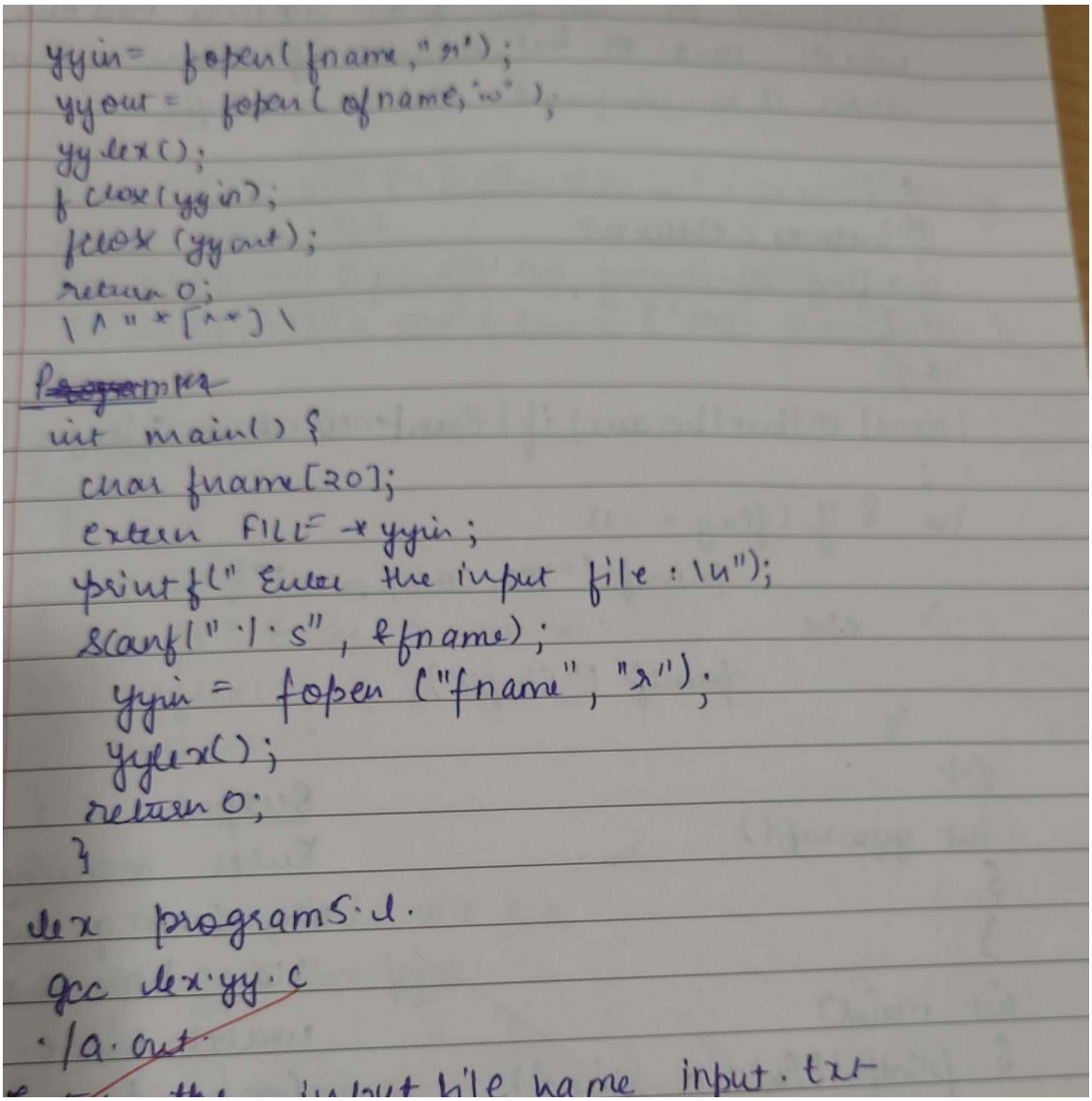
27/4/23

Output


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



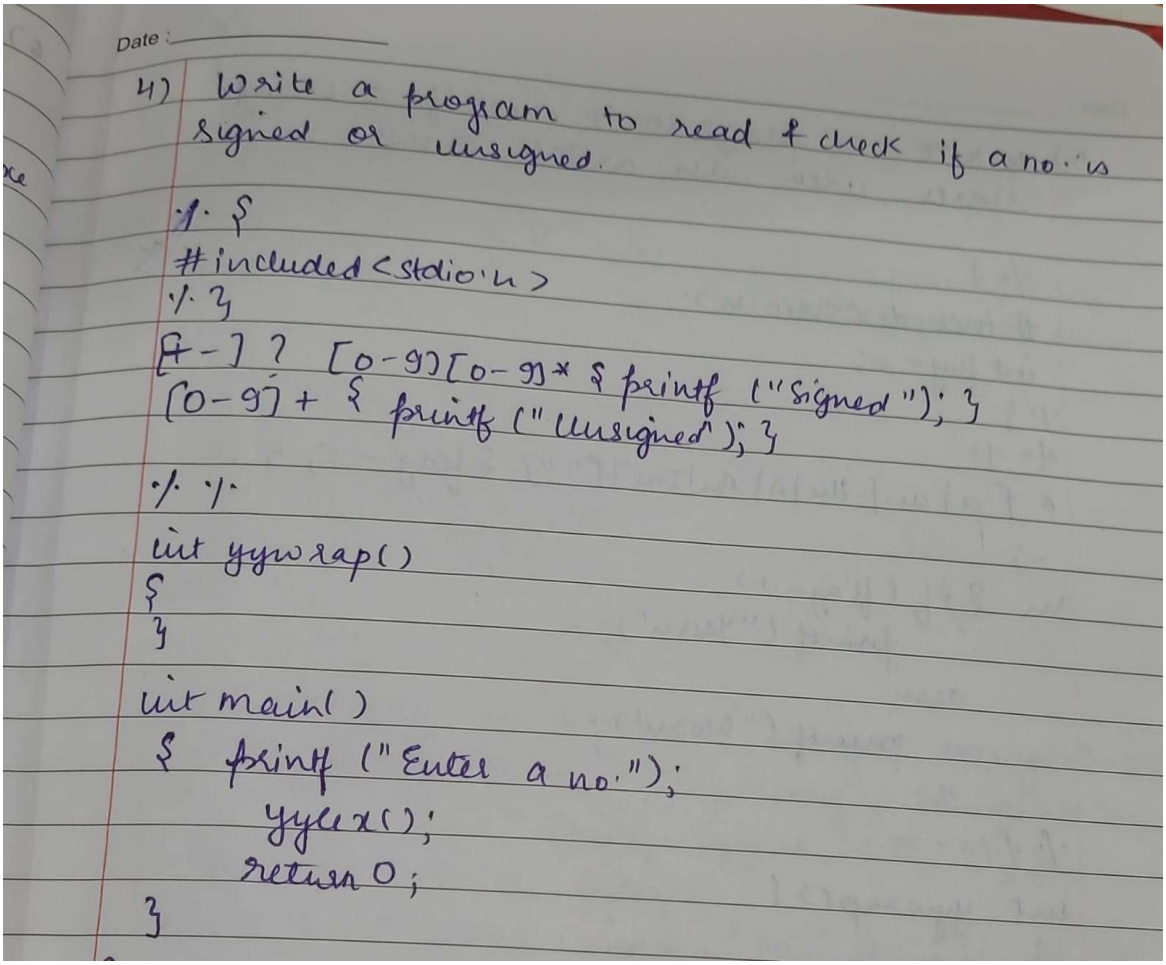
```
yyin= fopen(fname,"r");
yyout = fopen(ofname,"w");
yylex();
fclose(yyin);
fclose(yyout);
return 0;
\/\*[^*]\
Program
int main() {
 char fname[20];
 extern FILE *yyin;
 printf(" Enter the input file :\n");
 scanf("%s", fname);
  yyin = fopen("fname", "r");
  yylex();
 return 0;
 }
lex programs.l
gcc lex.yy.c
./a.out
```

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



```
Date :

4) Write a program to read & check if a no. is
   signed or unsigned.

%{
#included<stdio.h>
%}
[+-]? [0-9][0-9]* { printf ("signed "); }
[0-9]+ { printf ("unsigned"); }

%%
int yywrap()
{
}

int main()
{ printf ("Enter a no.");
    yylex();
    return 0;
}
```

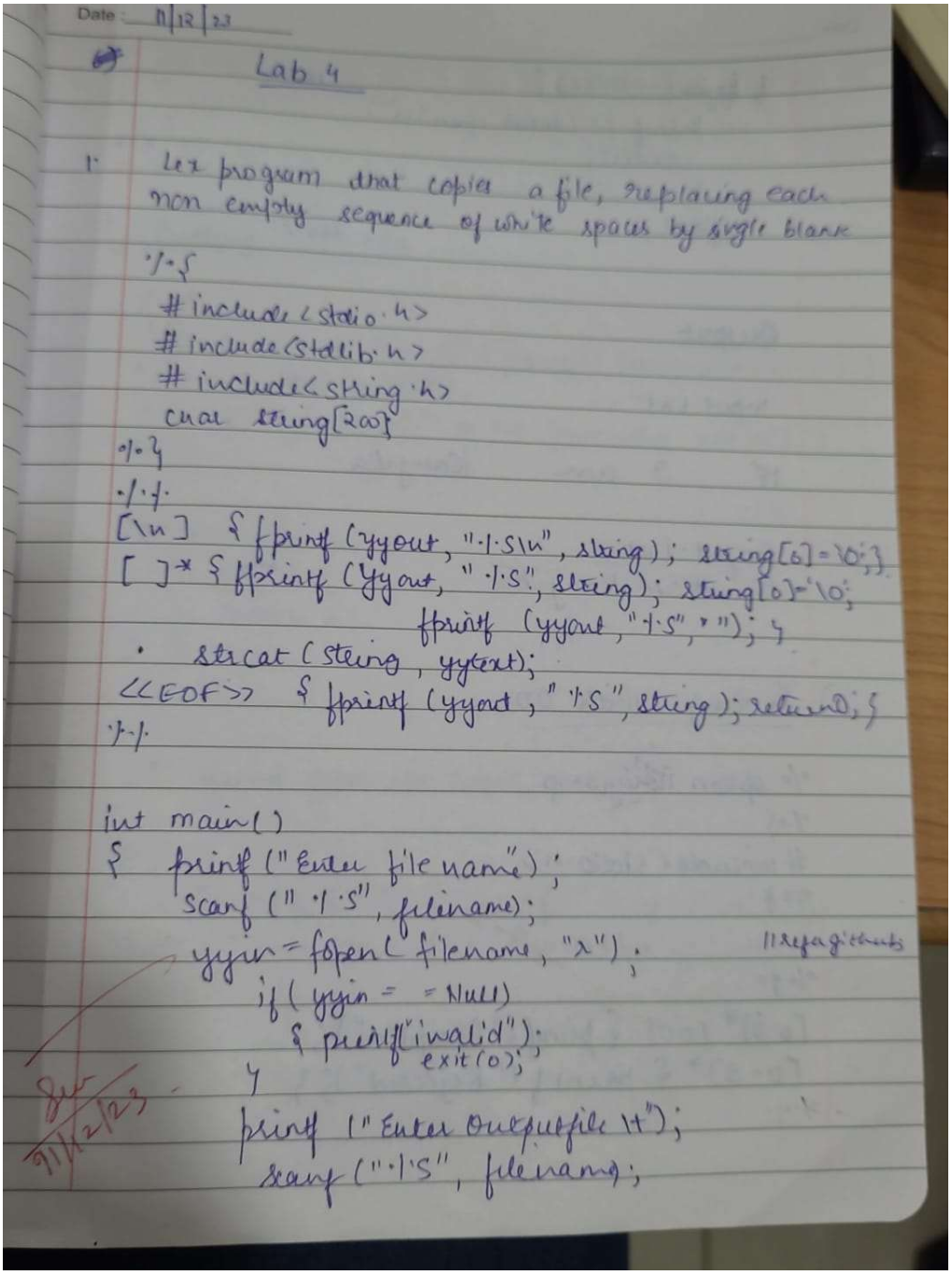
Output

```
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



Date : 11/12/23

Lab 4

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

```
%{
#include<stdio.h>
#include<stdlib.h>
#include<string.h>
char string[200];
%}
%%
[\n] { fprintf(yyout,"%s\n",string); string[0]='\0';}
[ ]* { fprintf(yyout," %s",string); string[0]='\0';
       fprintf(yyout,"%s"," "); }
. strcat(string, yytext);
<<EOF>> { fprintf(yyout," %s",string); return 0; }
%%

int main()
{ printf("Enter file name");
  scanf("%s",filename);
  yyin=fopen(filename,"r");          //regular expression
    if(yyin==NULL)
     { printf("invalid");
       exit(0);
     }
  printf("Enter Outputfile name");
  scanf("%s",filename);
```

9/12/23

Output


*text.txt
1 Hello World
2 Welcome to programming

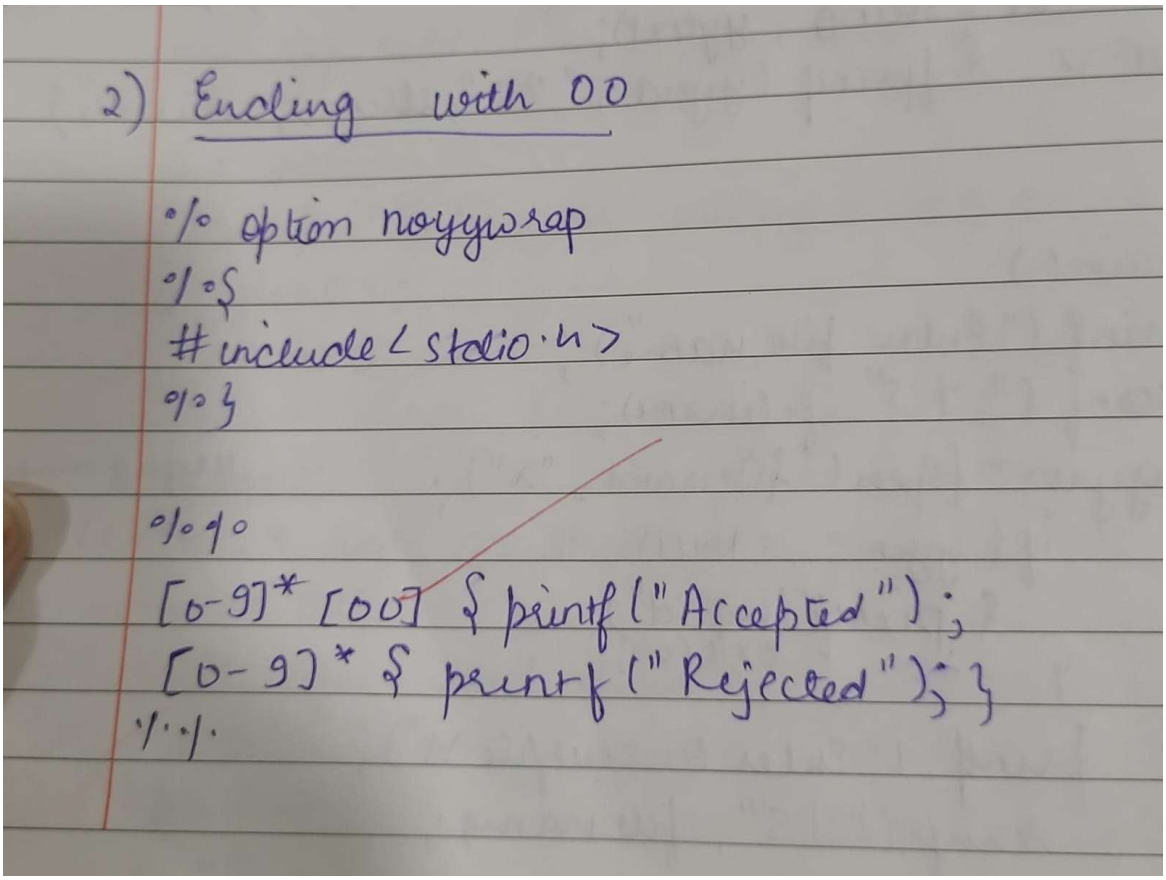

Printed!


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

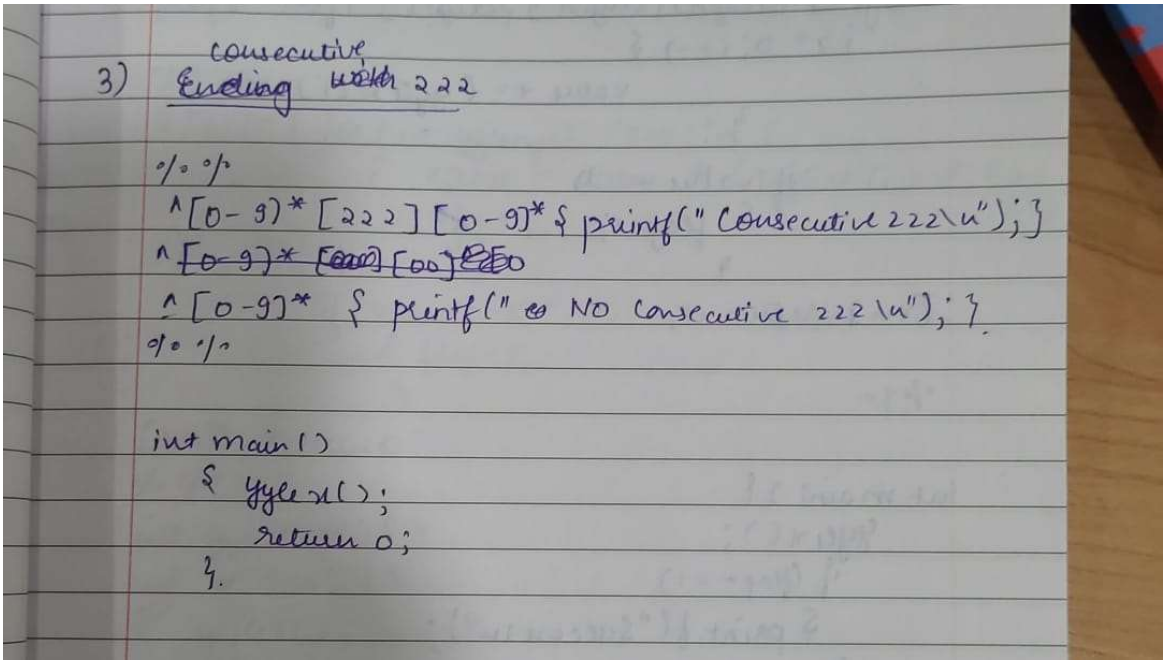


Output

```
neha29@neha-VirtualBox:~/Documents$ lex Week4_EndsWith0.l
neha29@neha-VirtualBox:~/Documents$ gcc lex.yy.c
neha29@neha-VirtualBox:~/Documents$ ./a.out
Enter a string:
12300
Ends with 0.
neha29@neha-VirtualBox:~/Documents$ gcc lex.yy.c
neha29@neha-VirtualBox:~/Documents$ ./a.out
Enter a string:
145
Does not end with 0.
neha29@neha-VirtualBox:~/Documents$
```

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

Code

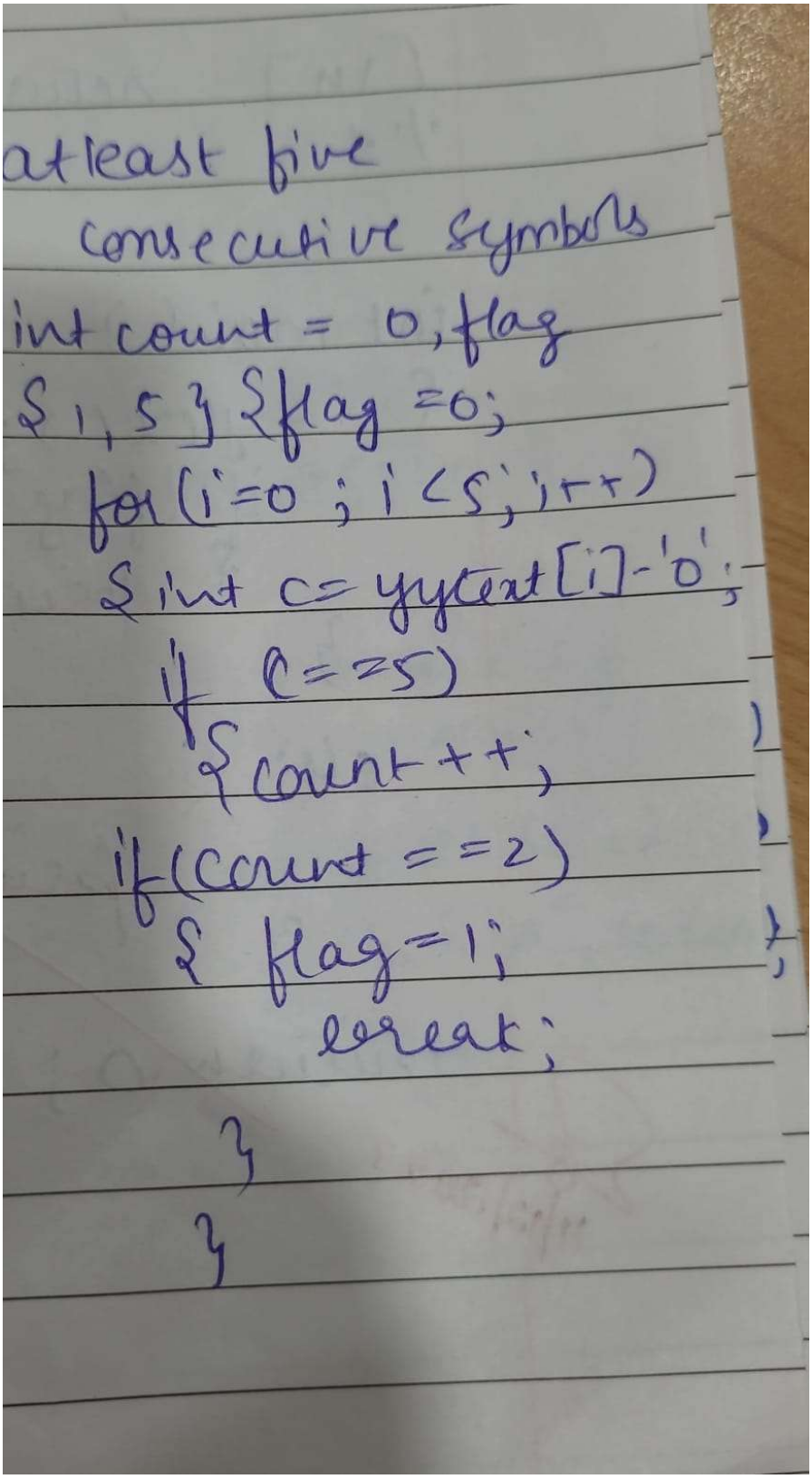


Output

```
neha29@neha-VirtualBox:~/Documents$ lex Week4_3Consecutive2s.l
neha29@neha-VirtualBox:~/Documents$ gcc lex.yy.c
neha29@neha-VirtualBox:~/Documents$ ./a.out
Enter a string:
2322
Does not have 3 consecutive 2's.
neha29@neha-VirtualBox:~/Documents$ gcc lex.yy.c
neha29@neha-VirtualBox:~/Documents$ ./a.out
Enter a string:
322221
Has 3 consecutive 2's.
neha29@neha-VirtualBox:~/Documents$
```

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

Code



```
atleast five
   consecutive symbols
int count = 0, flag
{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;
   }
}
```

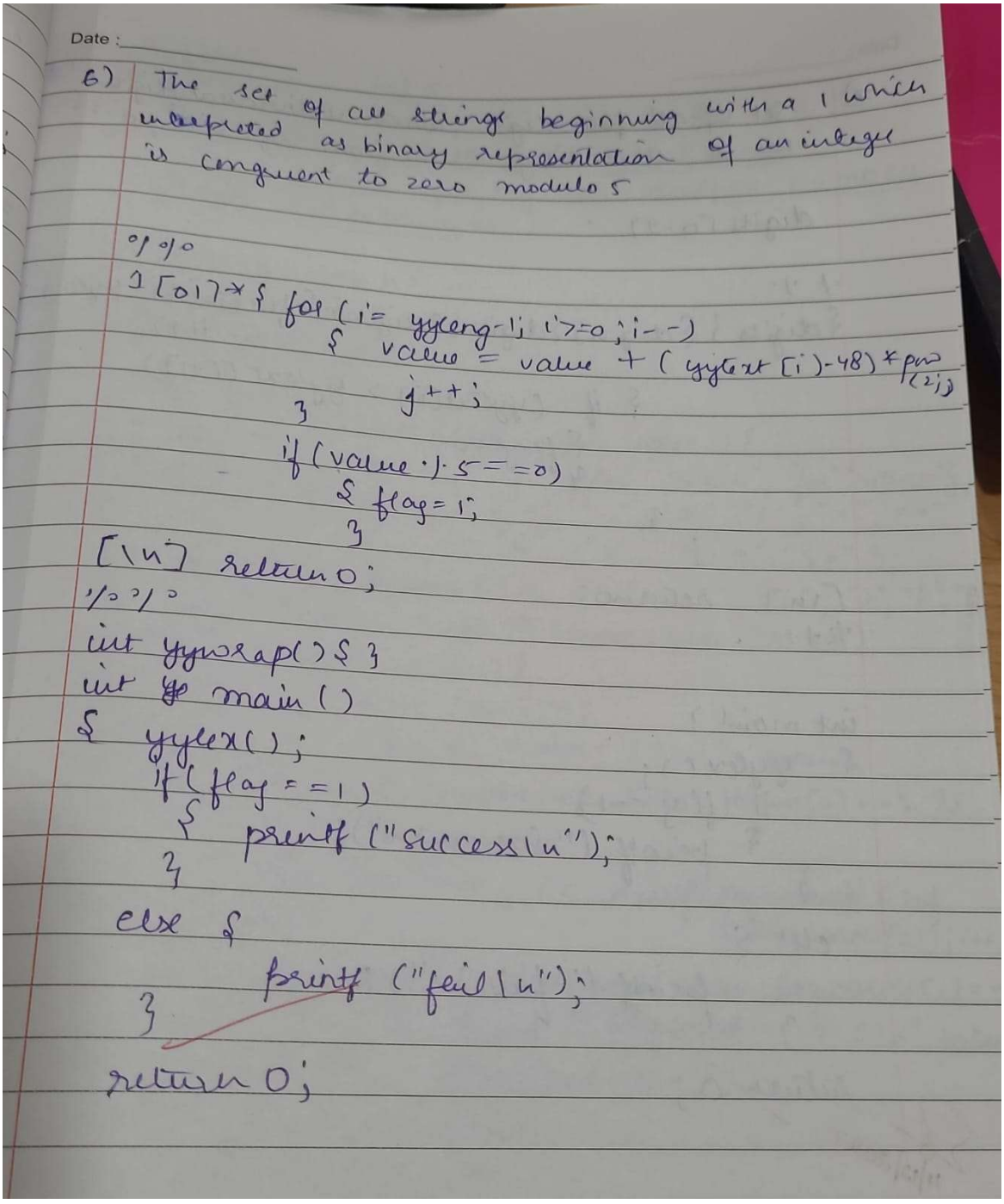
Output

```
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:
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!
```

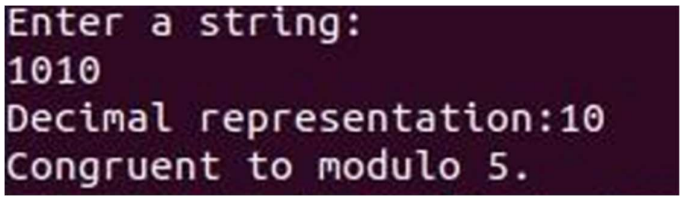
4.2.4 The set of all strings beginning with a 1 which, interpreted as the binary representation of an integer, is congruent to zero modulo 5.

Code



```
Date :
6) The set of all strings beginning with a 1 which
   interpreted as binary representation of an integer
   is congruent to zero modulo 5

%%%
1[01]* { for (i= yyleng-1; i>=0; i--)
          { value = value + (yytext[i]-48) * pow(2,j);
            j++;
          }
         if (value%5==0)
          { flag=1;
          }
[\n] return 0;
%%%
int yywrap() { }
int main()
{ yylex();
  if(flag==1)
   { printf("success\n");
   }
  else {
        printf("fail\n");
  }
return 0;
```

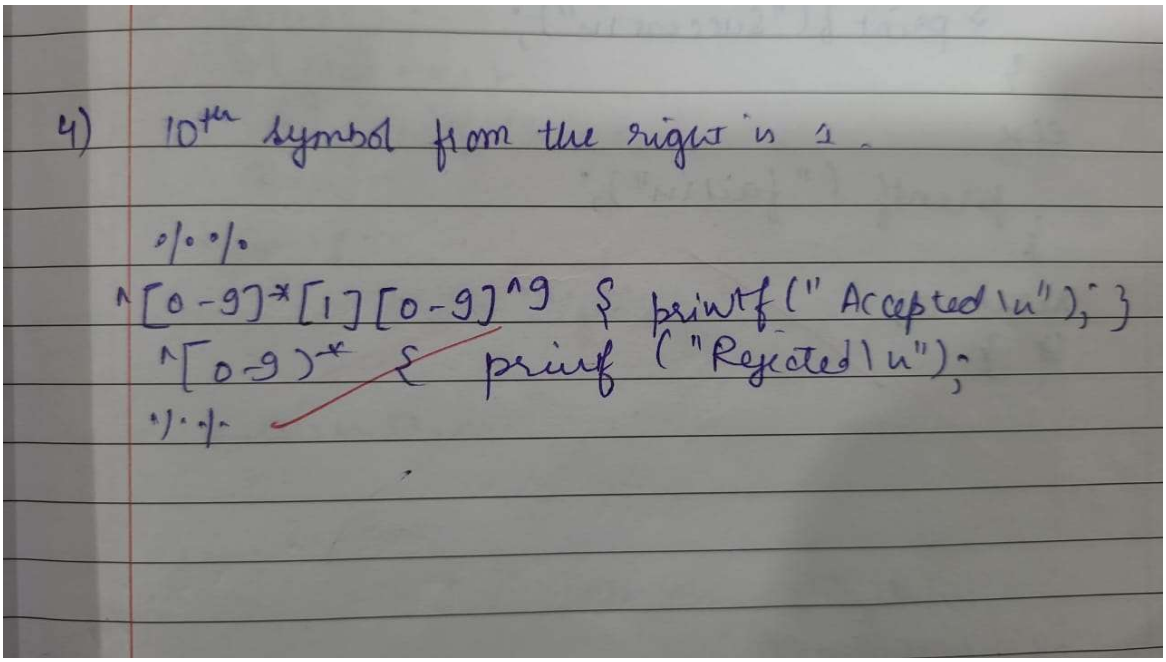
Output


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


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

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

Code



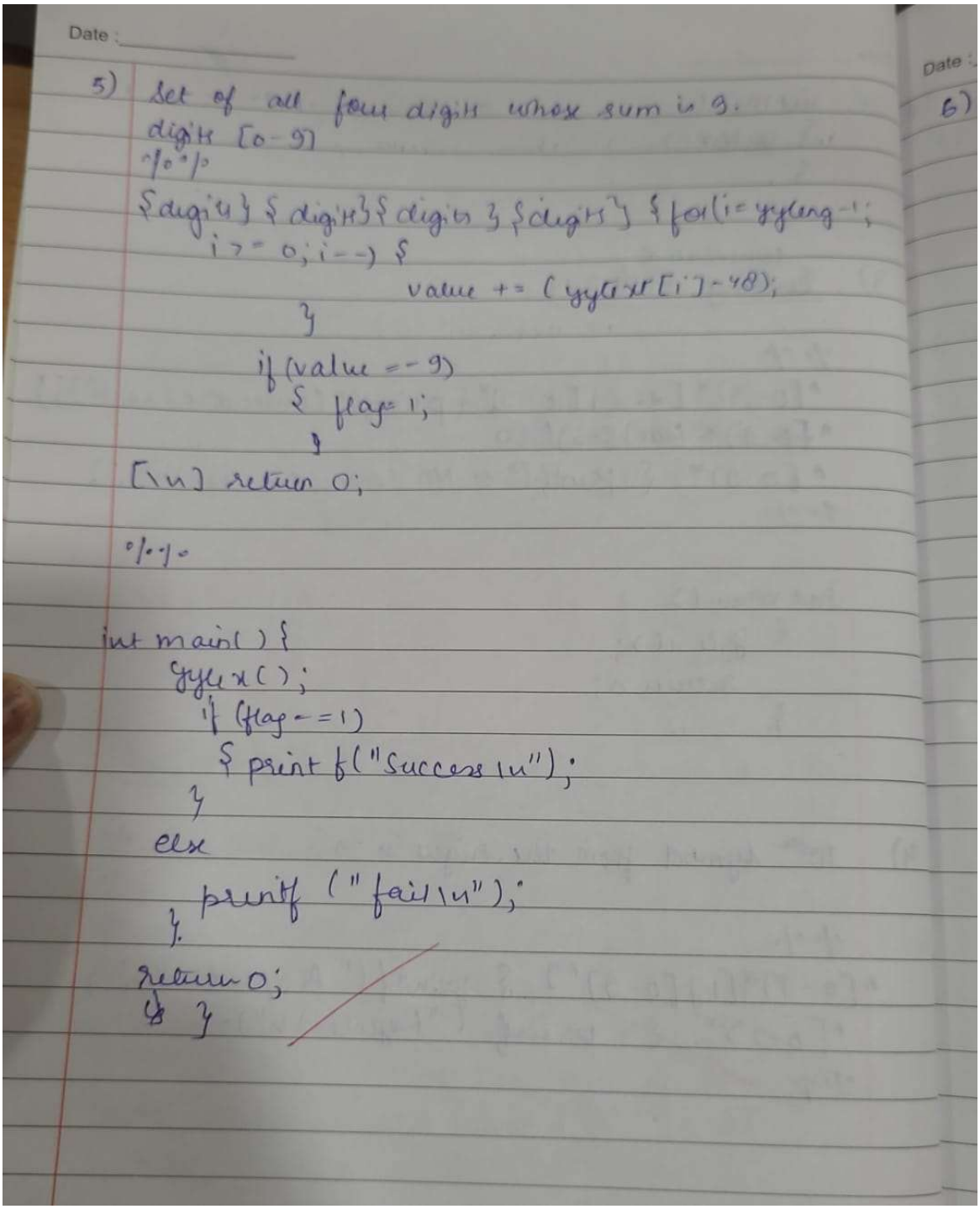
Output

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

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

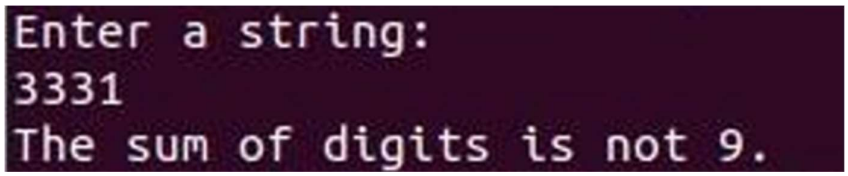
4.2.6 The set of all four digits numbers whose sum is 9.

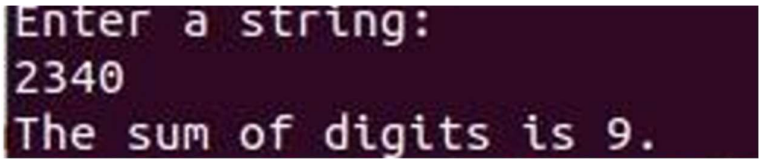
Code



```
5) Set of all four digits whose sum is 9.
digits [0-9]
%o%
${digits}${digits}${digits}${digits}$for(i=yylong-1;
    i>=0;i--){
            value += (yytext[i]-48);
        }
    if(value==9)
      { flag=1;
      }
[\n] return 0;

%o%

int main(){
    yylex();
     if(flag==1)
       $printf("Success\n");
    }
   else
     printf("fail\n");
   }
  return 0;
 }
```

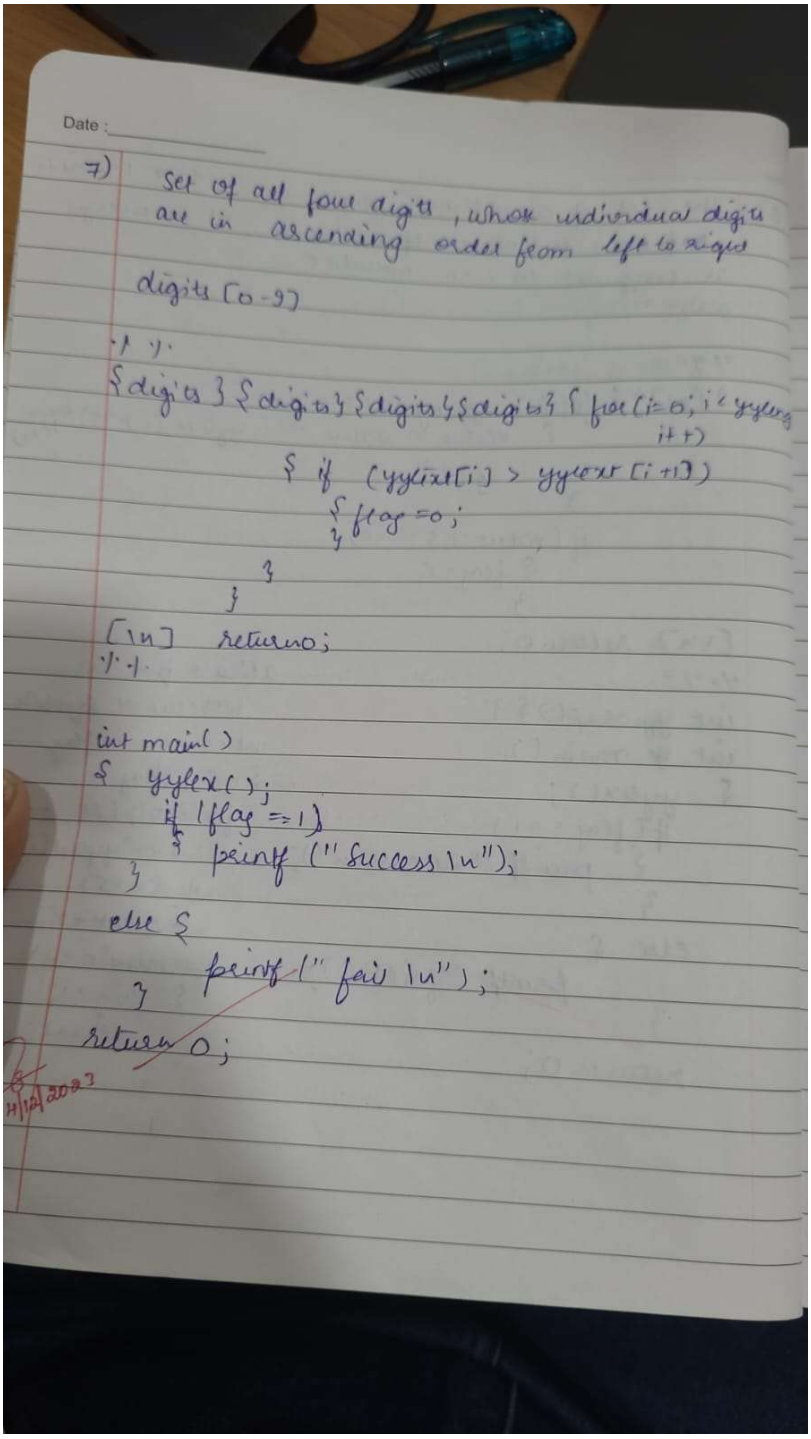
Output


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

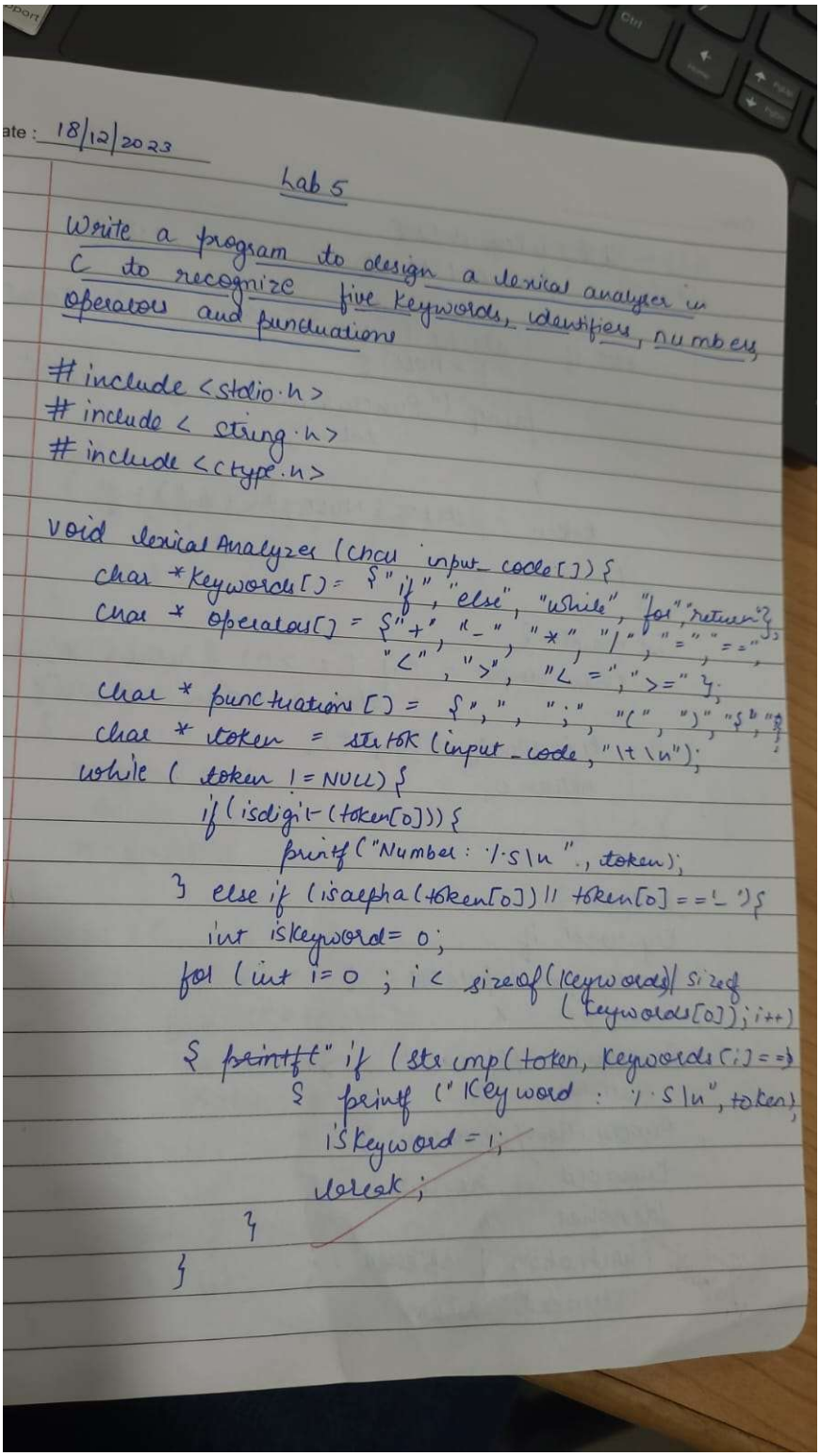
```
Enter a string:
1235
The digits are in ascending order.
```

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

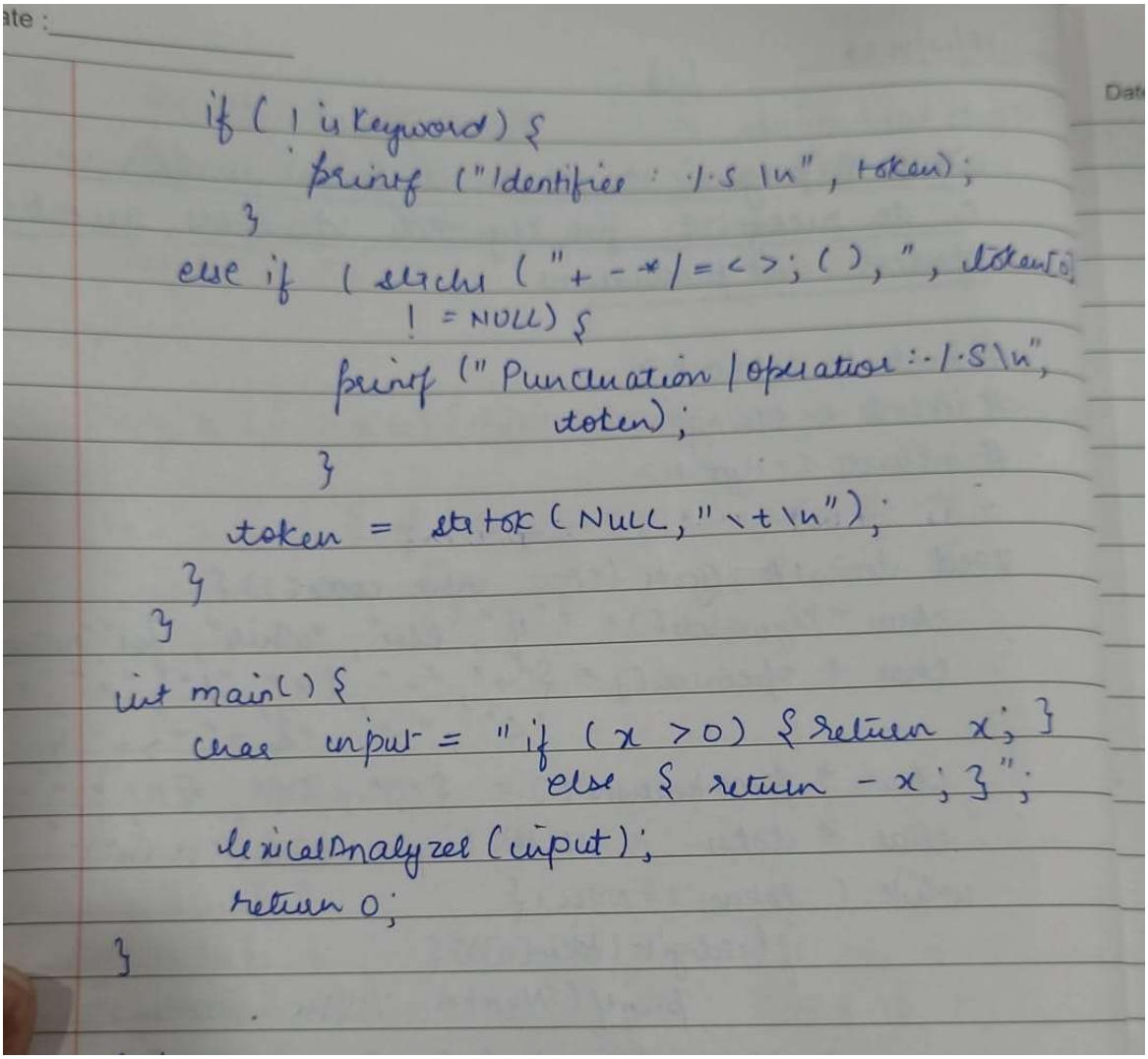
Lab 5

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

Code



```
Date : 18/12/2023

Lab 5

Write a program to design a lexical analyzer in
C to recognize five keywords, identifiers, numbers,
operators and punctuations

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

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



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

Output

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

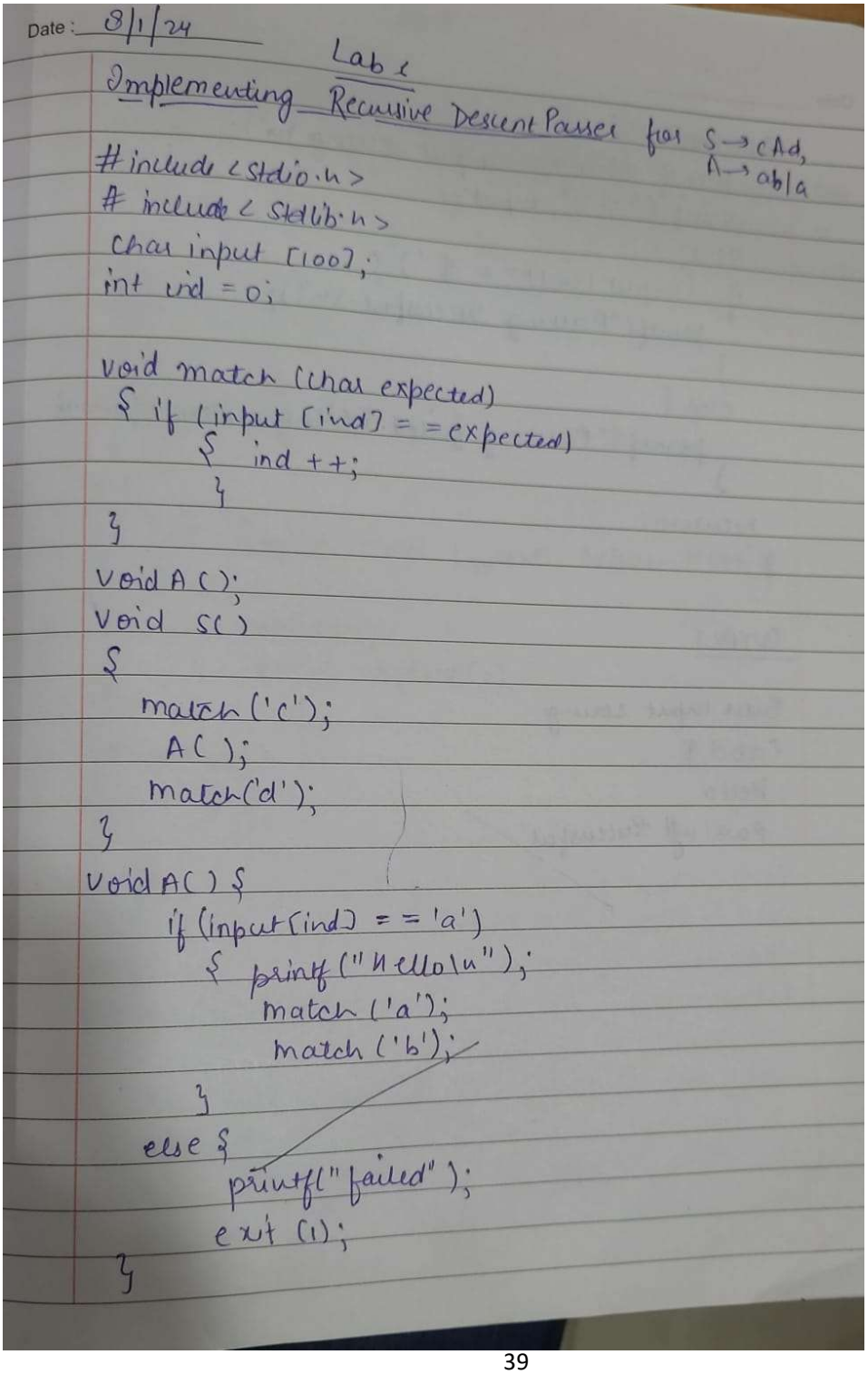
Lab 6

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

S->cAd

A->ab | a

Code

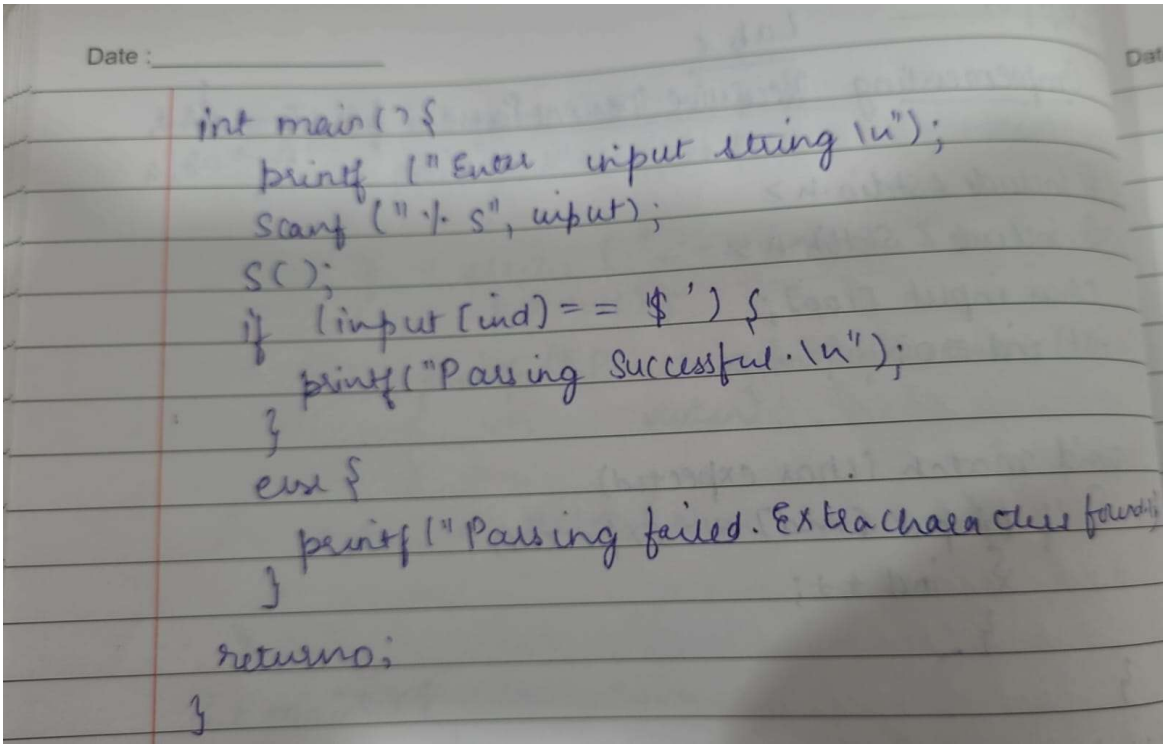


Date : 8/1/24

Lab 6

Implementing Recursive Descent Parser for S → cAd, A → ab|a

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

void match(char expected)
{ if (input[ind] == expected)
    { ind++;
    }
}
void A();
void S()
{
  match('c');
  A();
  match('d');
}
void A() {
   if (input[ind] == 'a')
     { printf("Hello\n");
       match('a');
       match('b');
     }
  else {
       printf("failed");
       exit(1);
}
```



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

Output

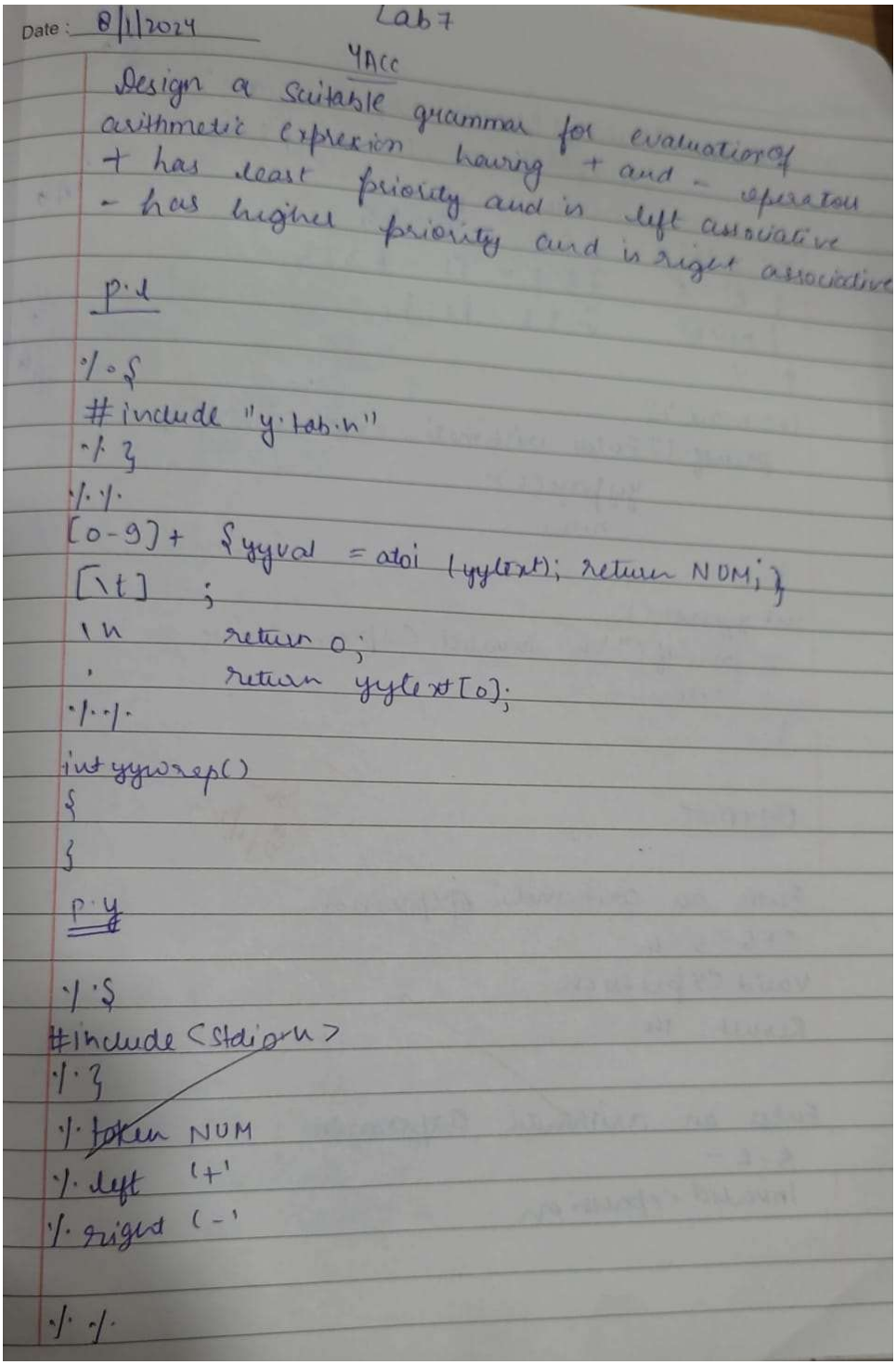
```
Enter a string:
cad$
Valid string!
```

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

Lab 7

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

Code



Date : 8/1/2024 Lab 7

YACC

Design a suitable grammar for evaluation of arithmetic expression having + and - operator
+ has least priority and is left associative
- has higher priority and is right associative

p.l

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

p.y

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

%%
```

Date :

```
exp : e { printf("valid expression\n");
          printf("Result : %d \n", $$);
          return 0; }
e : e '+' e    { $$ = $1 + $3; }
  | e '-' e    { $$ = $1 - $3; }
  | NUM        { $$ = $1; };
%%
int main(){
  printf("Enter arithmetic expression\n");
          yyparse();
          return 0;
}
int yyerror()
{ printf("\n Invalid expression\n");
    return 0;
}
```

OUTPUT

Enter an arithmetic expression
5+6-3-6
Valid expression
Result : 14.

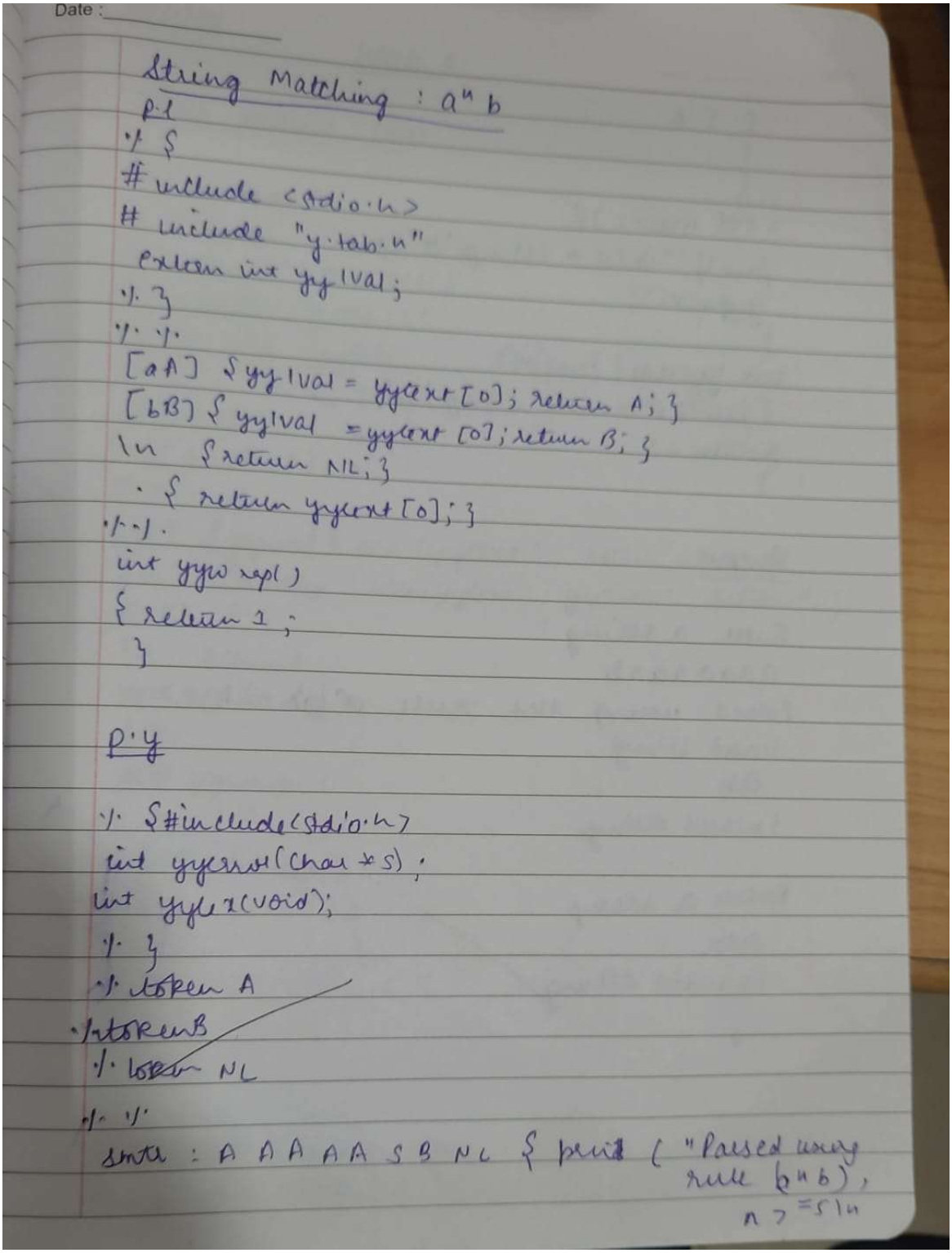
Enter an arithmetic expression
5-6-
Invalid expression

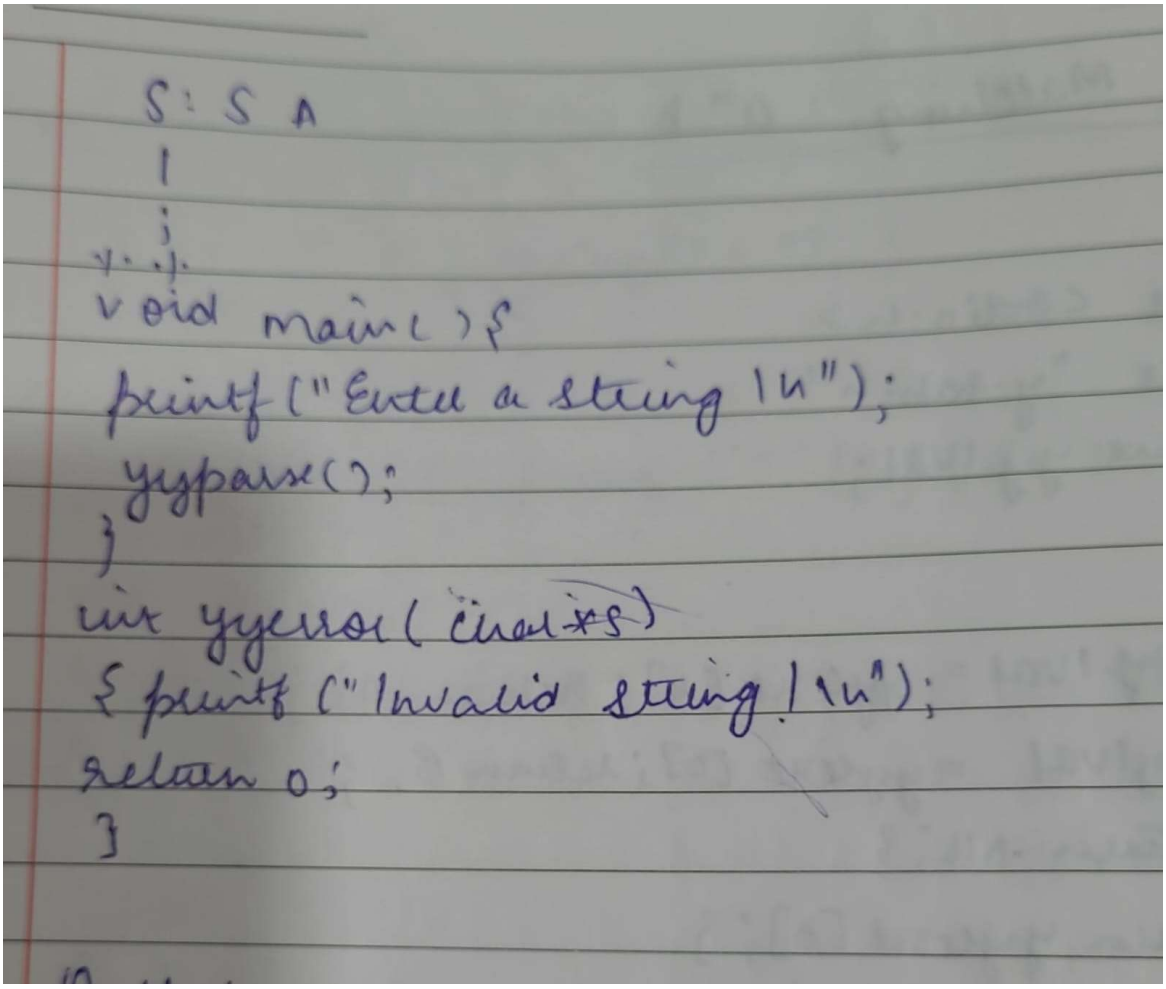
Output

```
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



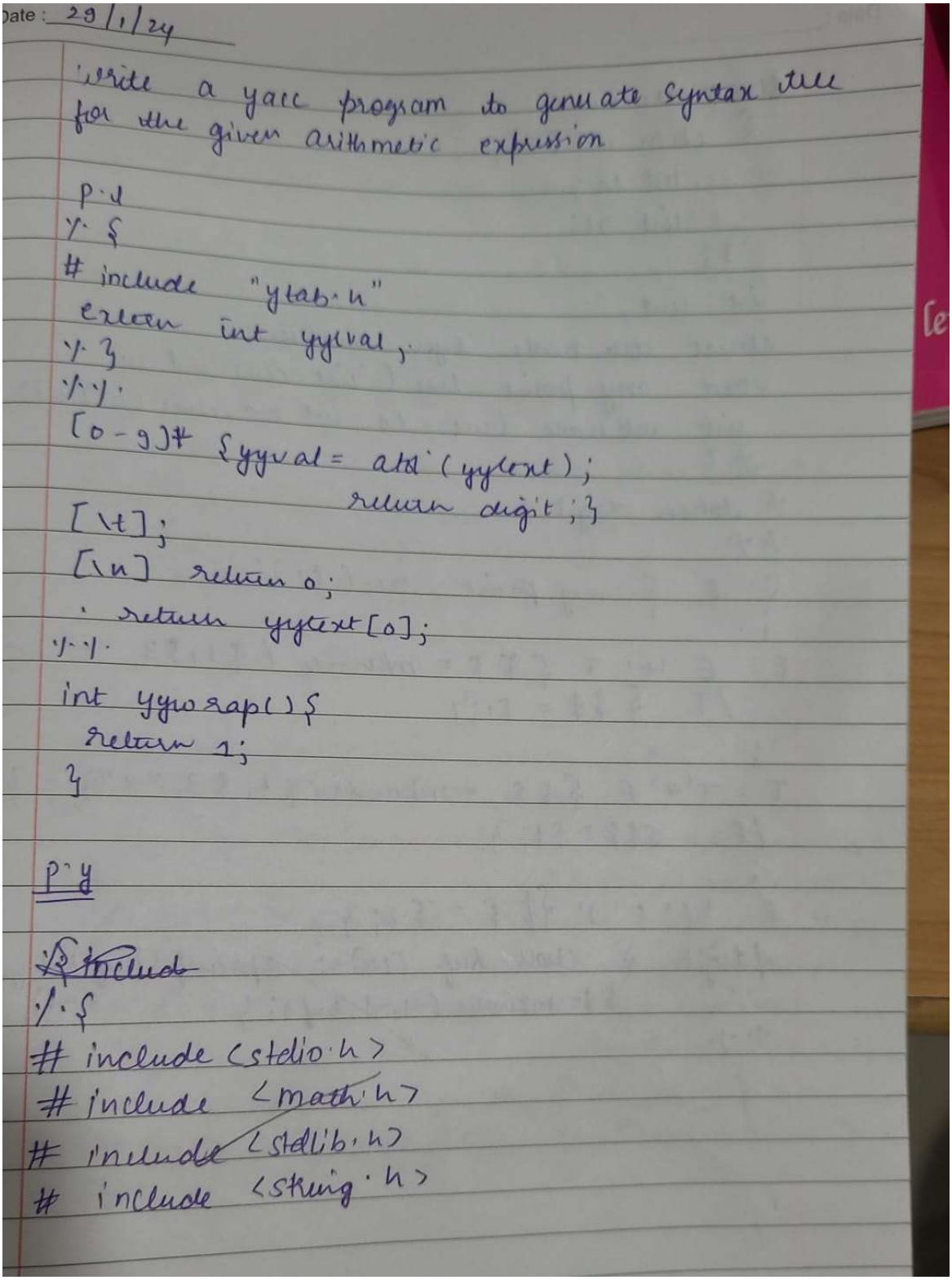


Output

```
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



Date : 29/1/24

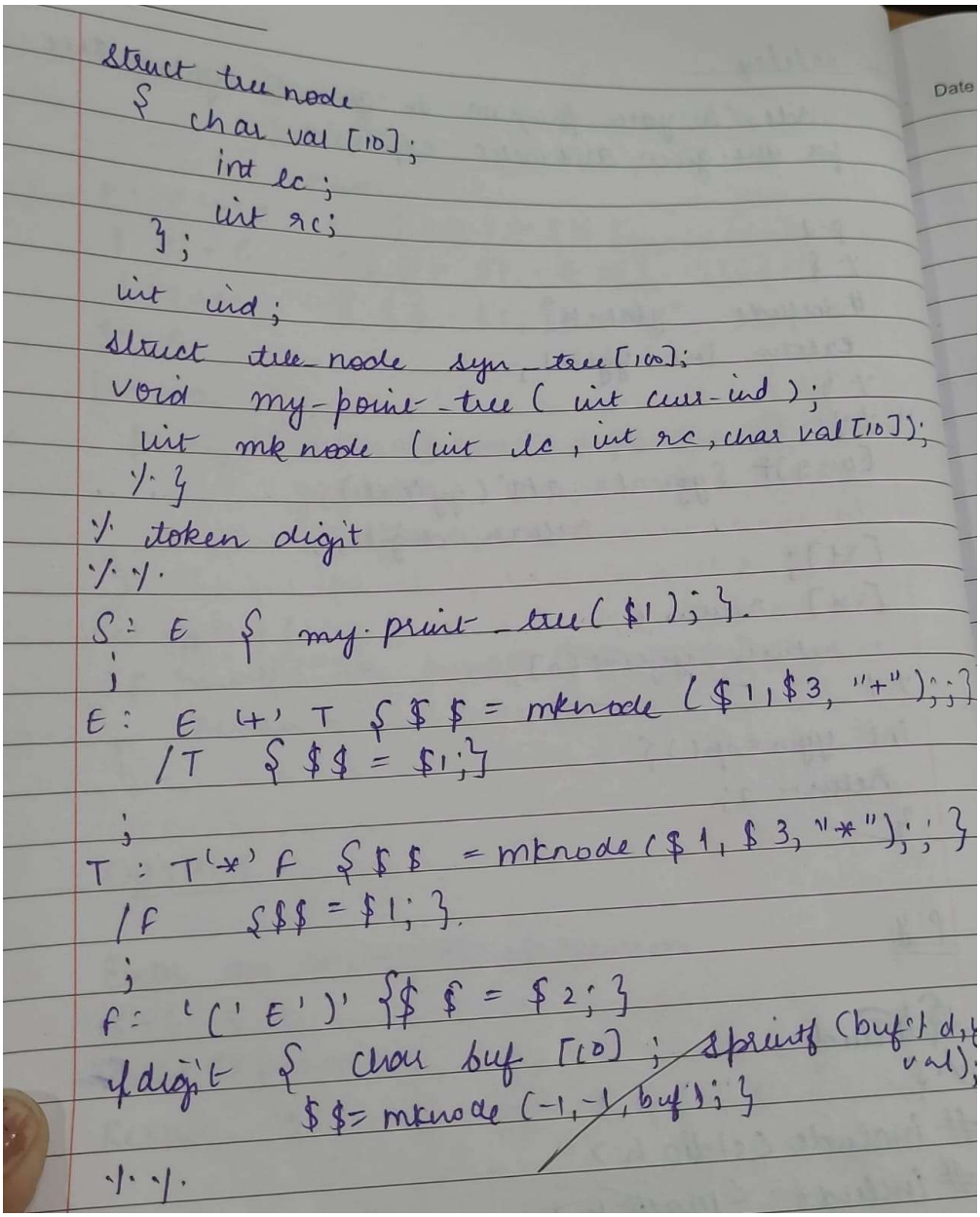
write a yacc program to generate syntax tree for the given arithmetic expression

p.l

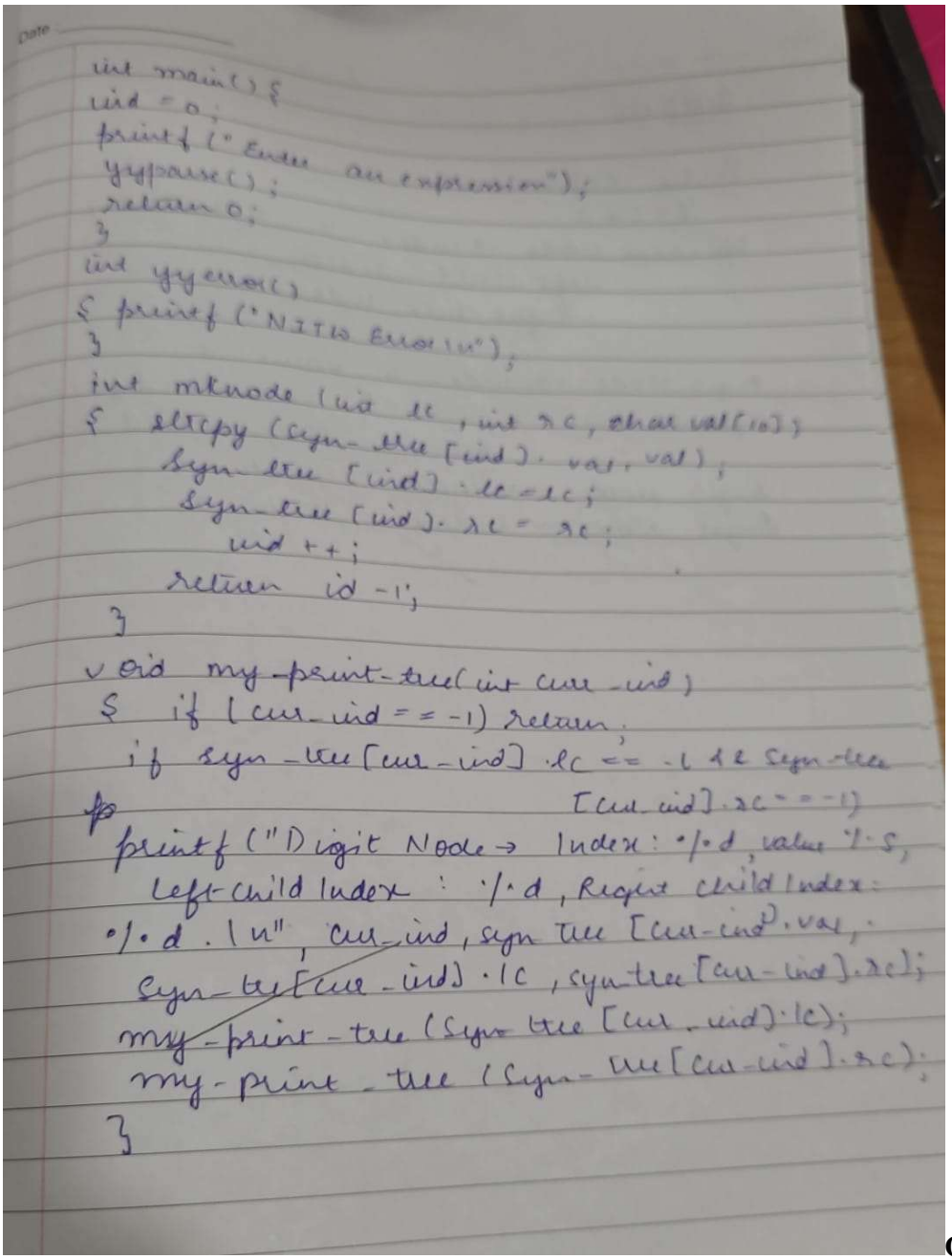
```
%{
#include "ytab.h"
extern int yylval;
%}
%%
[0-9]+ {yylval = atoi(yytext);
        return digit;}
[\t];
[\n] return 0;
. return yytext[0];
%%
int yywrap(){
 return 1;
}
```

p.y

```
%{
#include <stdio.h>
#include <math.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 ')' { $$ = $2; }
digit { char buf[10]; sprintf (buf, "%d",
                                val);
        $$ = mknode (-1, -1, buf); }
%. %.
```



```
int main() {
uid = 0;
printf("Enter an expression");
yyparse();
return 0;
}
int yyerror()
{ printf("NITK Error\n");
}
int mknode(int lc, int rc, char val[10]);
{ strcpy(syn_tree[uid].var, val);
  syn_tree[uid].lc = lc;
  syn_tree[uid].rc = rc;
    uid++;
  return id-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,
  Left child Index : %d, Right child Index:
%d.\n", cur_ind, syn_tree[cur_ind].var,
  syn_tree[cur_ind].lc, syn_tree[cur_ind].rc);
my_print_tree(syn_tree[cur_ind].lc);
my_print_tree(syn_tree[cur_ind].rc);
}
```

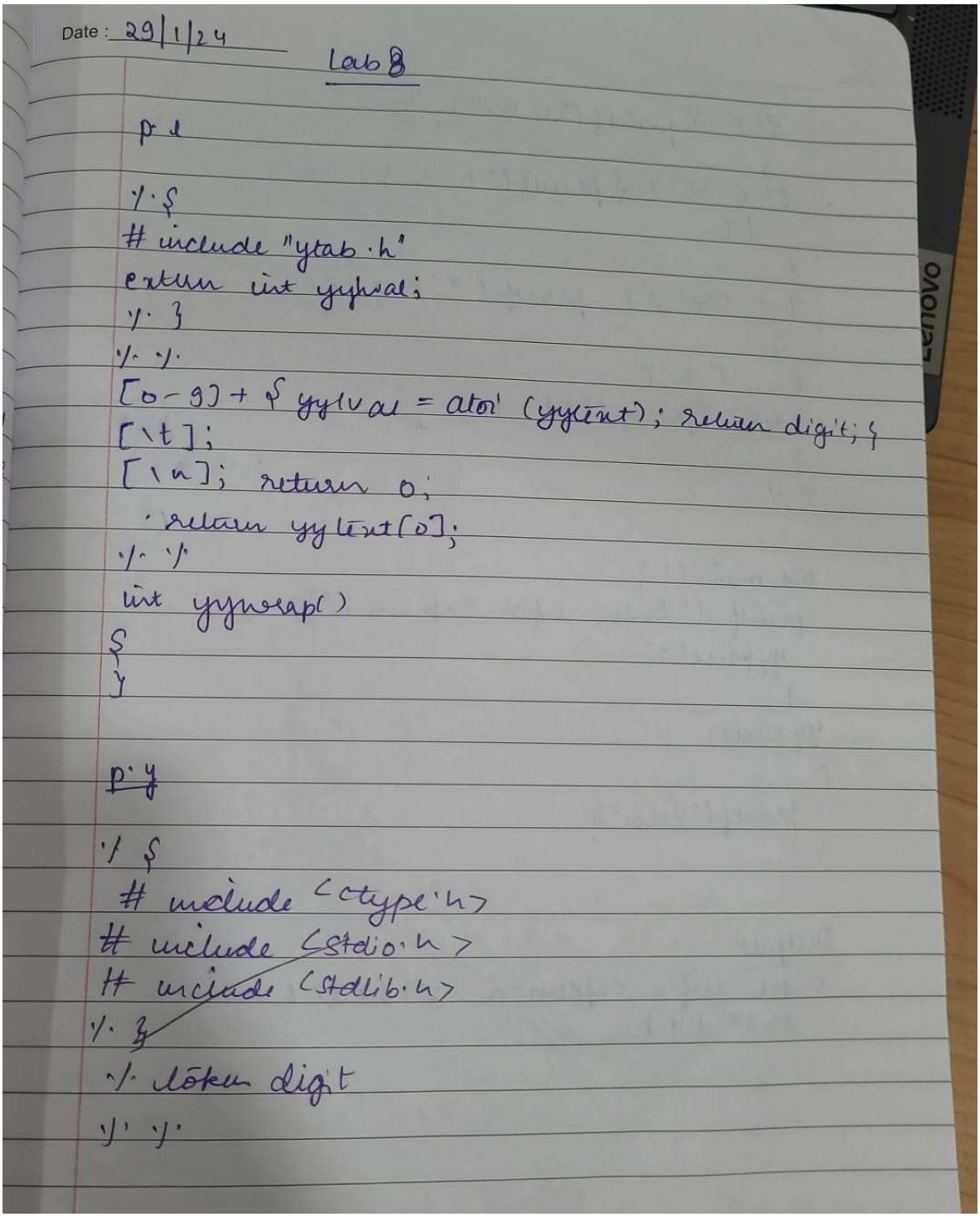
Output

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



```
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 infix expression 2+6*3+4

263* +4+

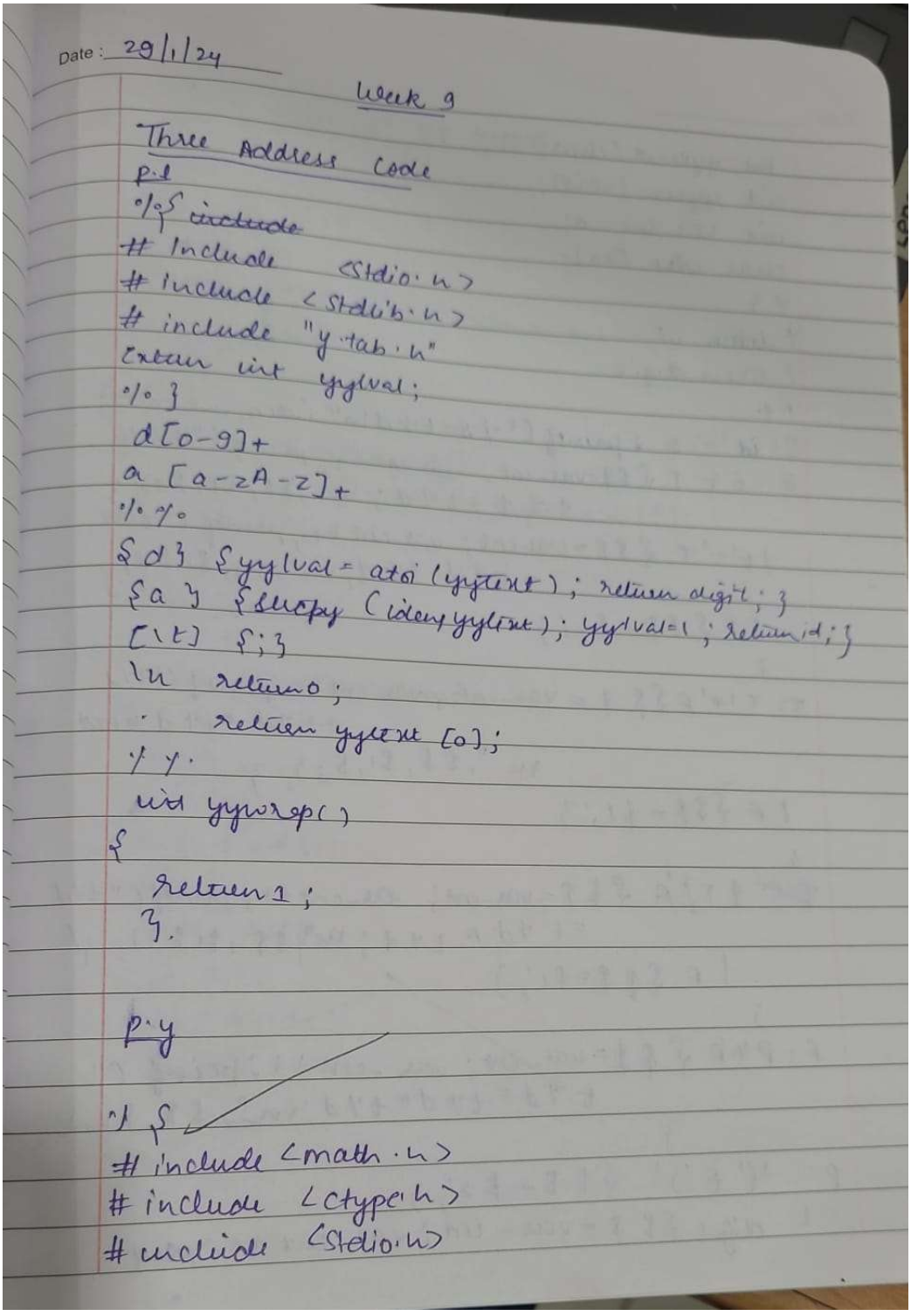
Output

```
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



Date : 29/1/24

Week 9

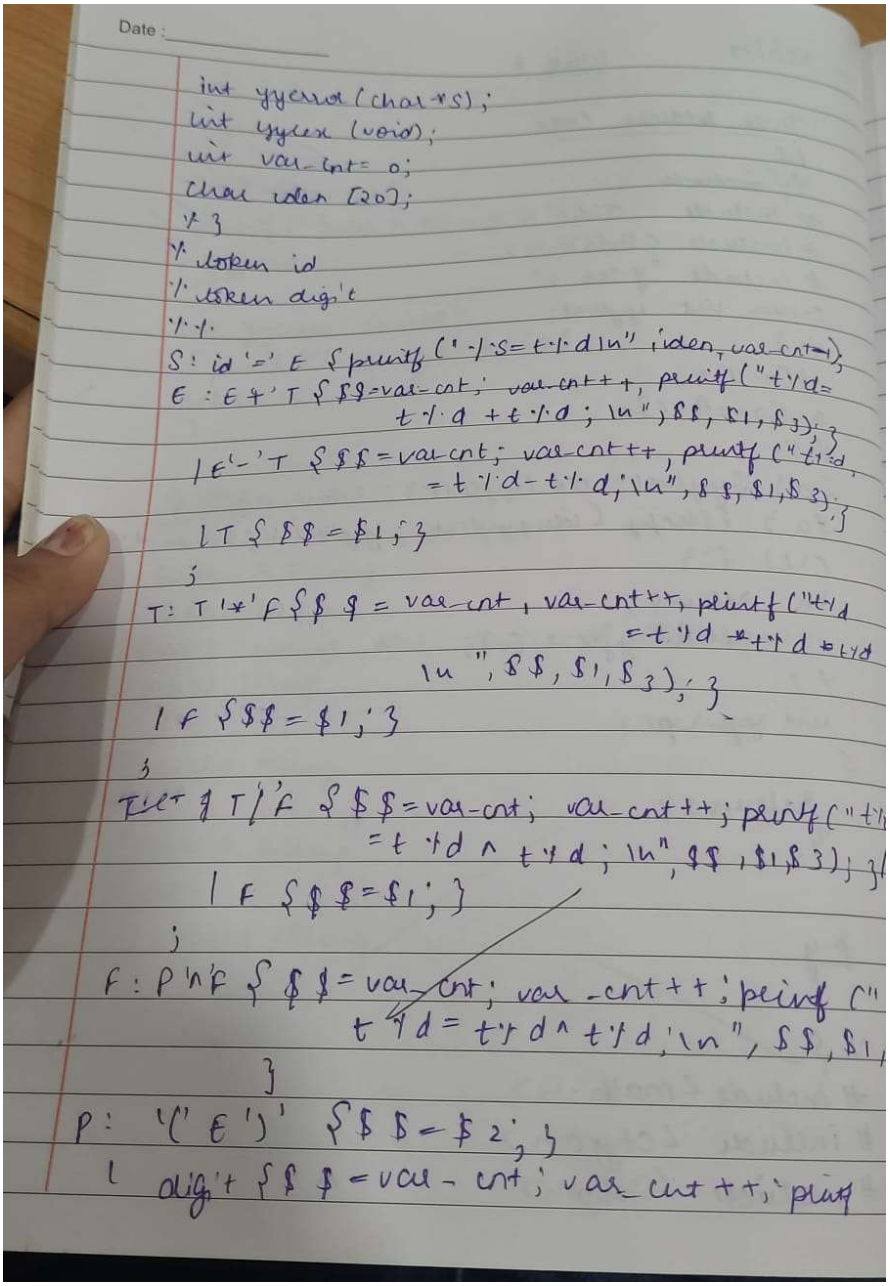
Three Address Code

p.l

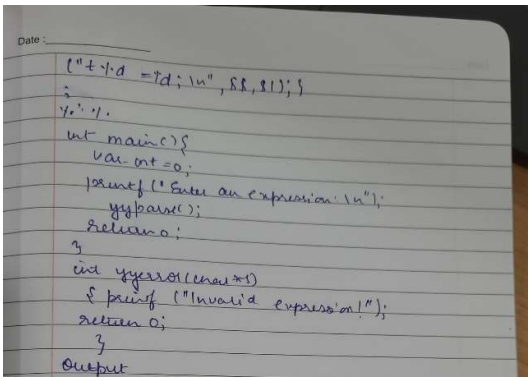
```
%{ include
# Include <stdio.h>
# include <stdlib.h>
# include "y.tab.h"
Extern int yylval;
%}
d[0-9]+
a [a-zA-Z]+
%%
{d} {yylval = atoi(yytext); return digit; }
{a} {strcpy (iden,yytext); yylval=1; return id;}
[\t] {;}
\n return 0;
. return yytext[0];
%%
int yywrap()
{
return 1;
}
```

p.y

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



```
Date :

int yyerror(char *s);
int yylex(void);
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 {$$ = var_cnt; var_cnt++; printf("t%d =
        t%d + t%d; \n", $$, $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", $$, $1, $3);}
 | F {$$ = $1;}
 ;
T: T '/' F {$$ = var_cnt; var_cnt++; printf("t%
        = t%d ^ t%d; \n", $$, $1, $3);}
 | F {$$ = $1;}
 ;
F: P '^' F {$$ = var_cnt; var_cnt++; printf("
        t%d = t%d ^ t%d; \n", $$, $1,
    }
P: '(' E ')' {$$ = $2;}
 | digit {$$ = var_cnt; var_cnt++; printf
```



```
Date :

("t%d = %d; \n", $$, $1);}
;
%%
int main(){
  var_cnt = 0;
  printf("Enter an expression: \n");
  yyparse();
  return 0;
}
int yyerror(char *s)
{ printf("Invalid expression!");
  return 0;
}
Output
```

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
```