VISVESVARAYA TECHNOLOGICAL UNIVERSITY

"JnanaSangama", Belgaum -590014, Karnataka.

LAB REPORT

on

COMPILER DESIGN

Submitted by

Gautam Deo (1BM21CS067)

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 **Gautam Deo (1BM21CS067)** , 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
Assistant professor
Department of CSE
BMSCE, Bengaluru

Dr. Jyothi Nayak
Professor and Head
Department of CSE
BMSCE, Bengaluru

B. M. S. COLLEGE OF ENGINEERING

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

DECLARATION

I, Gautam Deo (1BM21CS067), student of 5th Semester, B.E, Department of Computer Science and Engineering, B. M. S. College of Engineering, Bangalore, here by declare that, this lab report entitled " **Compiler Design**" has been carried out by me under the guidance of Prof. Sunayana S, Assistant Professor, Department of CSE, B. M. S. College of Engineering, Bangalore during the academic semester November-2023-February-2024.

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

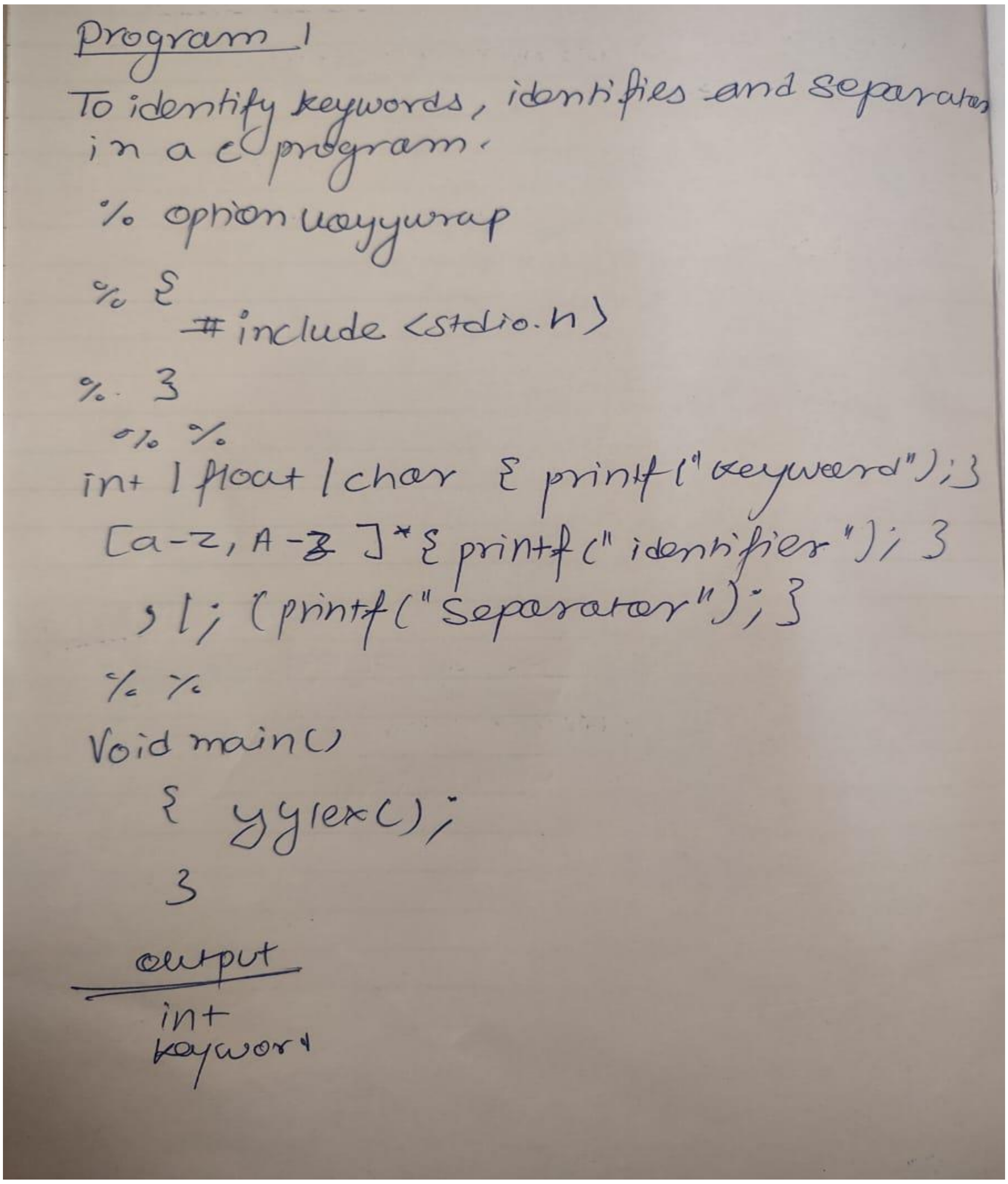
TABLE OF CONTENTS

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

Lab 1

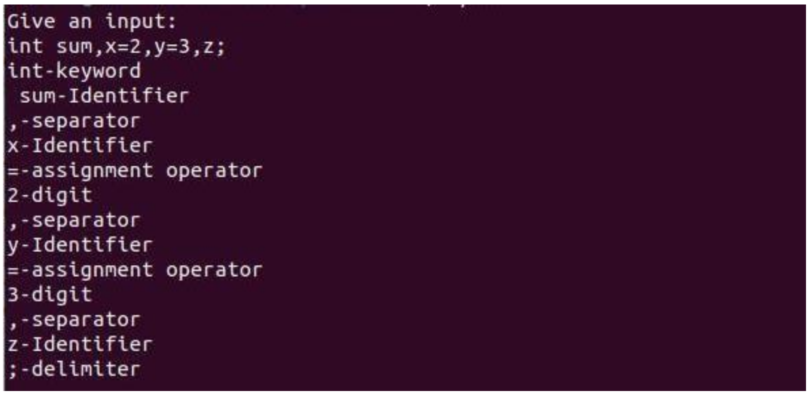
1.1 Write a program in LEX to recognize different tokens: Keywords, Identifiers, Constants, Operators and Punctuation symbols.

Code:



```
Program 1
To identify keywords, identifies and separaters
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();
   }
   output
   int
   keyword
```

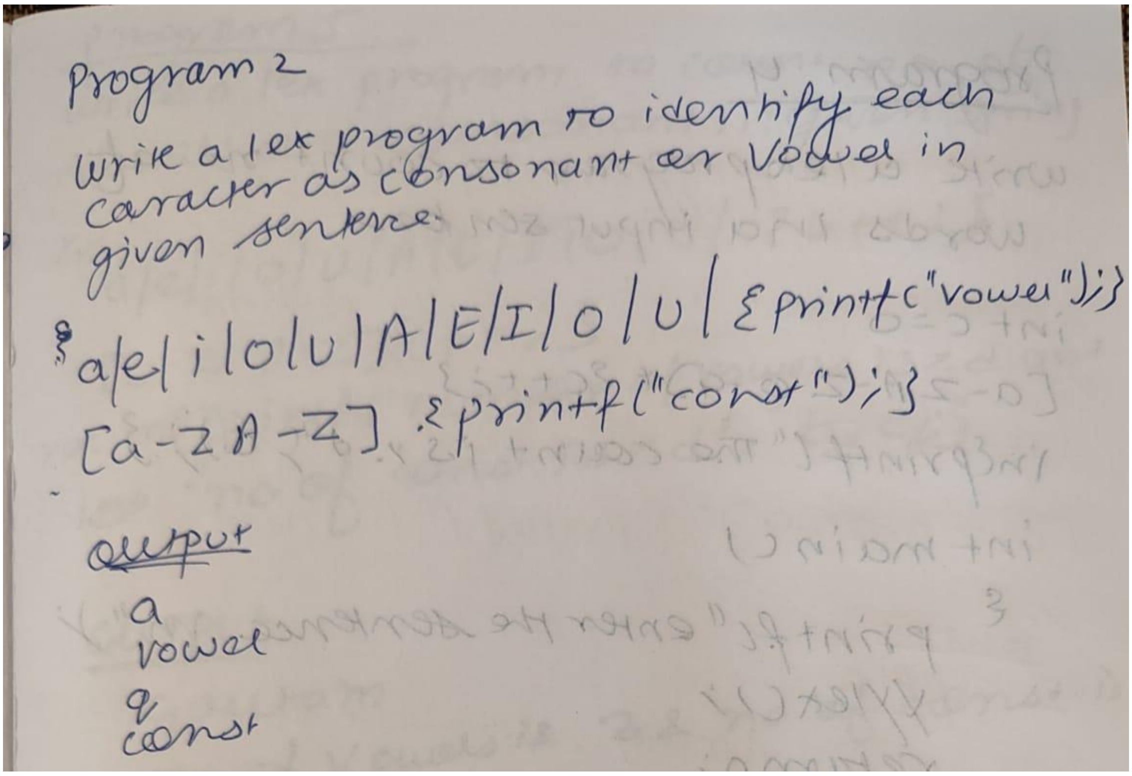
Output



```
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 rite a program in LEX to count the number of vowels and consonants in a string.

Code



Program 2

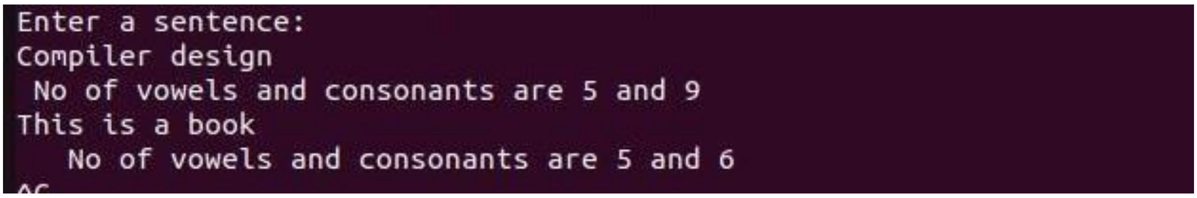
Write a lex program to identify each caracter as consonant or Vowel in given sentence.

```
%
a|e|i|o|u|A|E|I|O|U| {printf("vowel");}
[a-zA-Z] {printf("const");}
```

output

```
a
vowel
q
const
```

Output

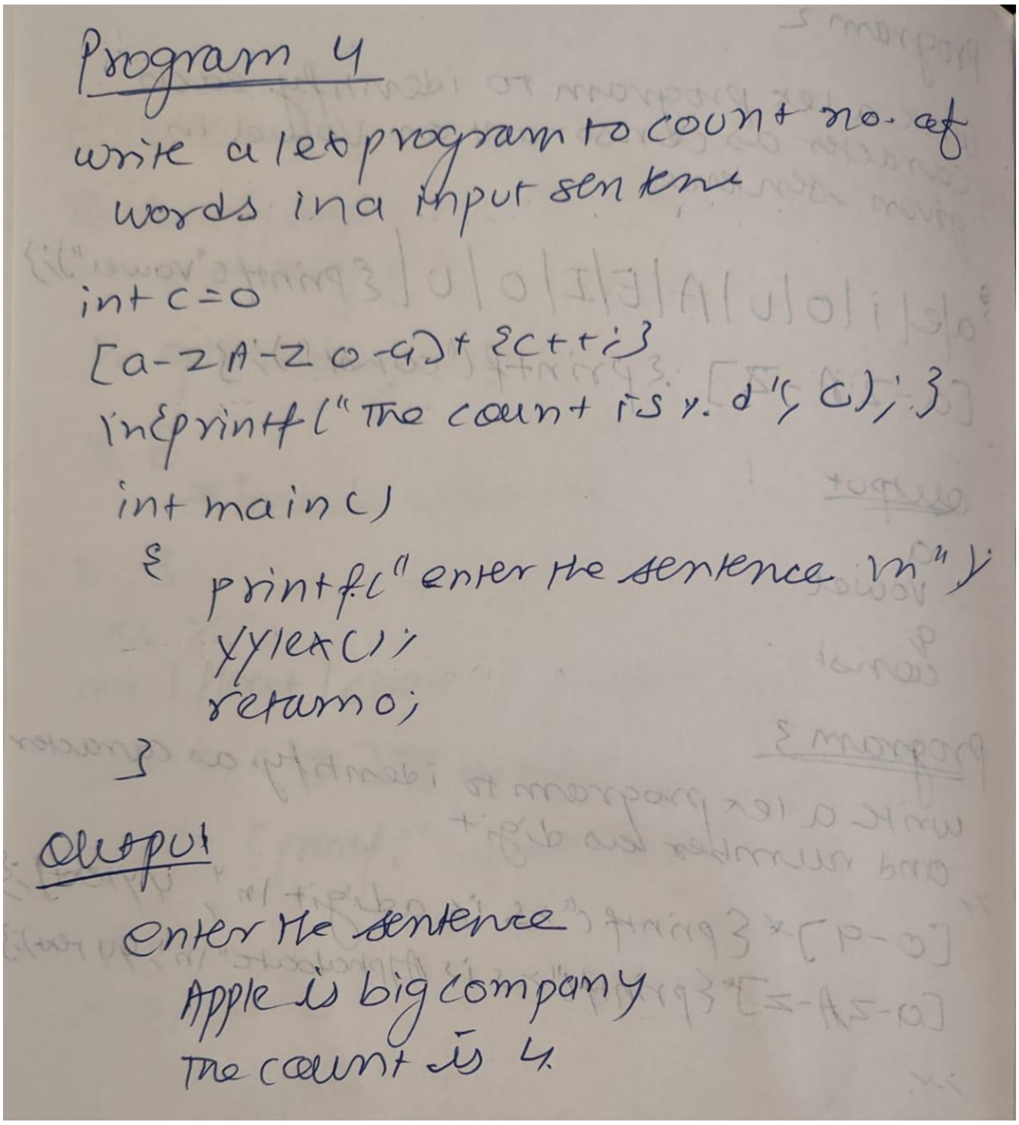


```
Enter a sentence:
Compiler design
 No of vowels and consonants are 5 and 9
This is a book
   No of vowels and consonants are 5 and 6
^C
```

Lab 2

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

Code



Program 4

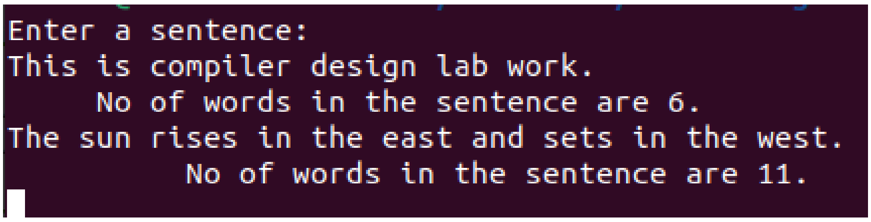
write a lex program to count no. of words ina input sentence

```
int c=0
[a-zA-Z 0-9]+ {c++;}
\n{printf("The count is %d", c);}
int main()
{
   printf("enter the sentence \n");
   yylex();
   return 0;
}
```

Output

```
enter the sentence
   Apple is big company
   The count is 4.
```

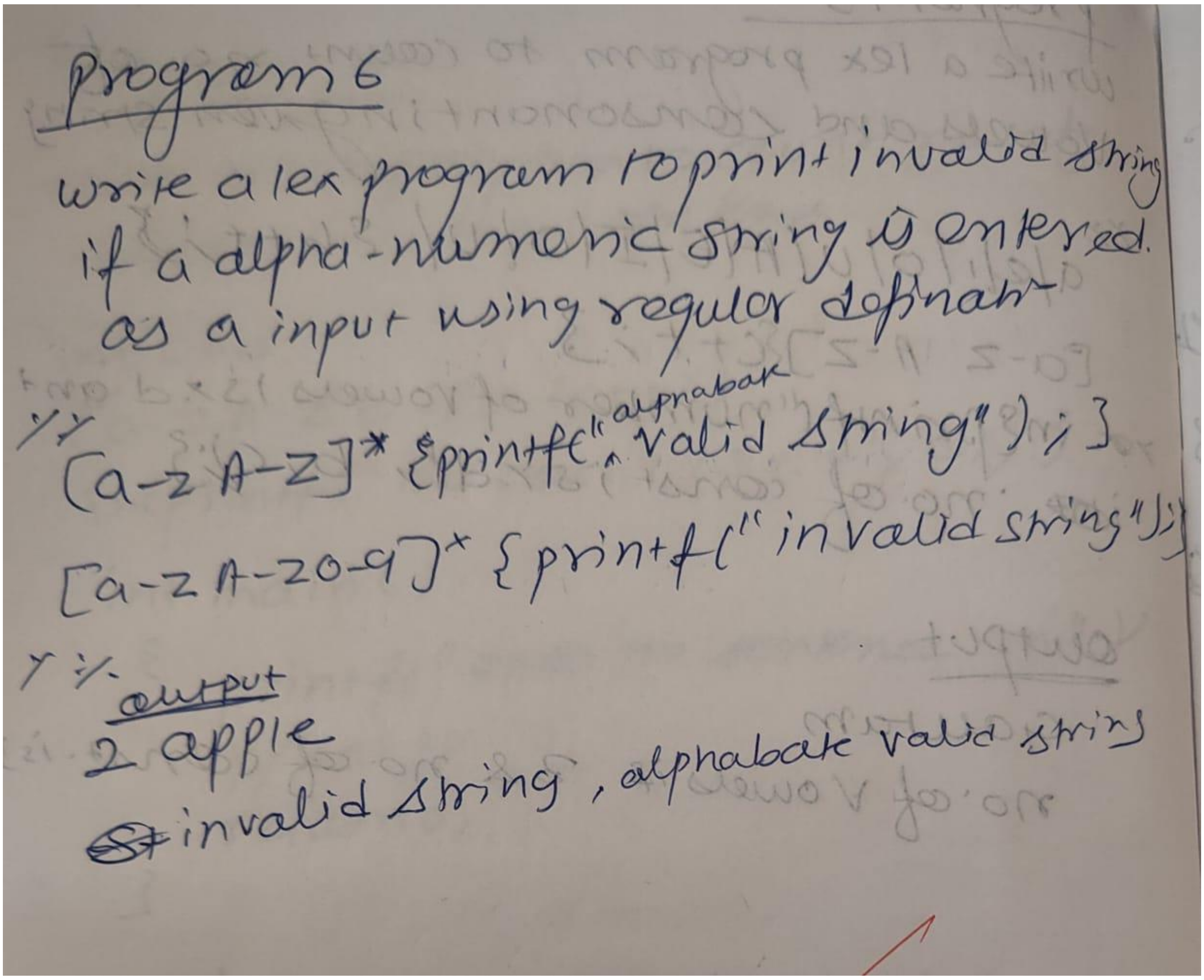
Output



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

2.2 Write a program in lex to demonstrate regular definition.

Code



Program 6

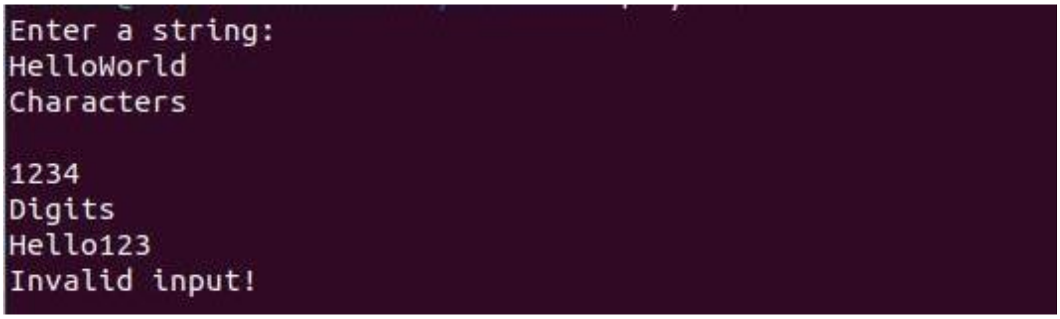
write a lex program to print invalid string if a alpha-numeric string is entered. as a input using regular defination

```
%%
[a-z A-Z]* {printf("alphabat valid string");}
[a-z A-Z 0-9]* {printf("invalid string");}
%%
```

output

```
2 apple
invalid string , alphabat valid string
```

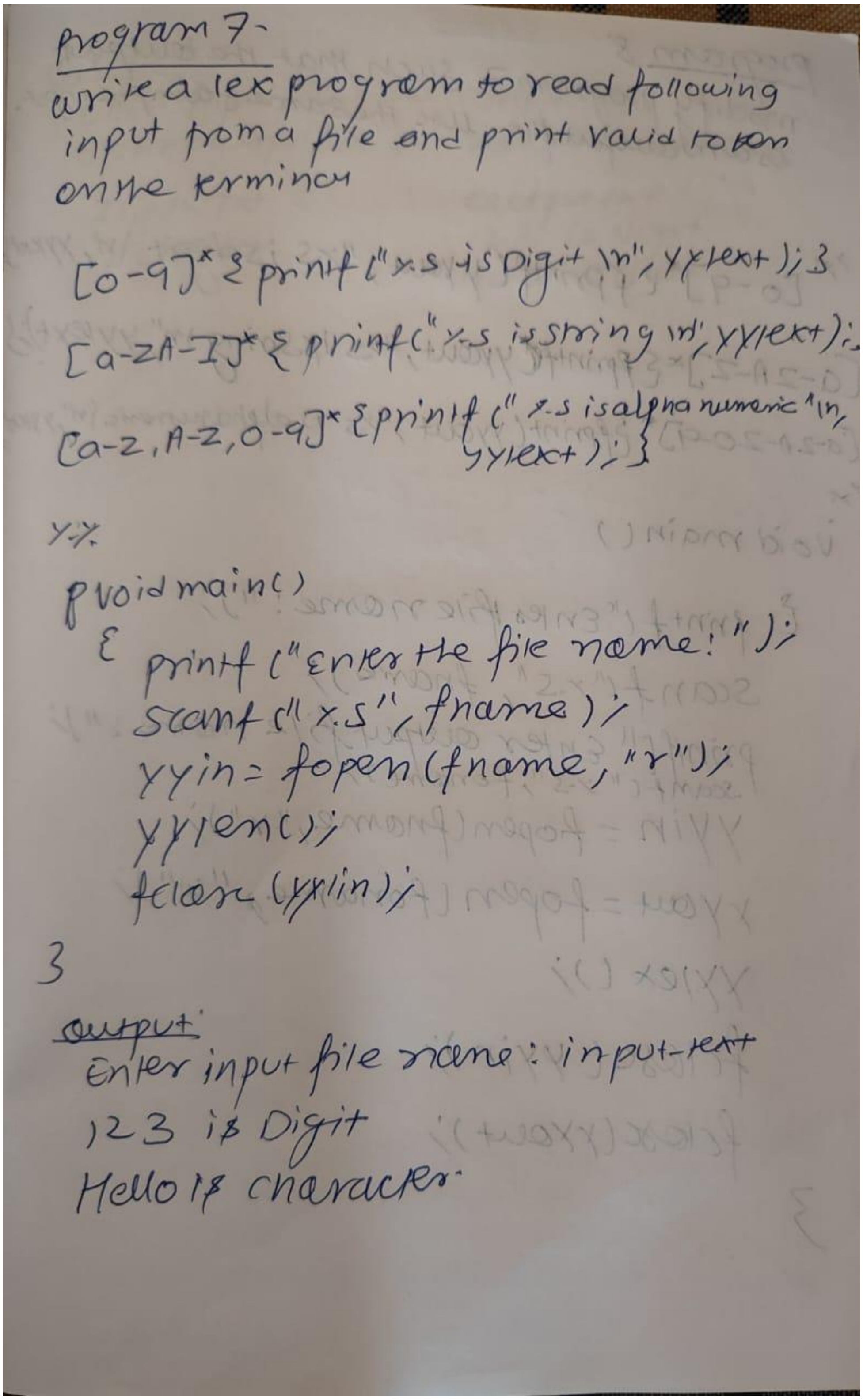
Output



```
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



program 7-

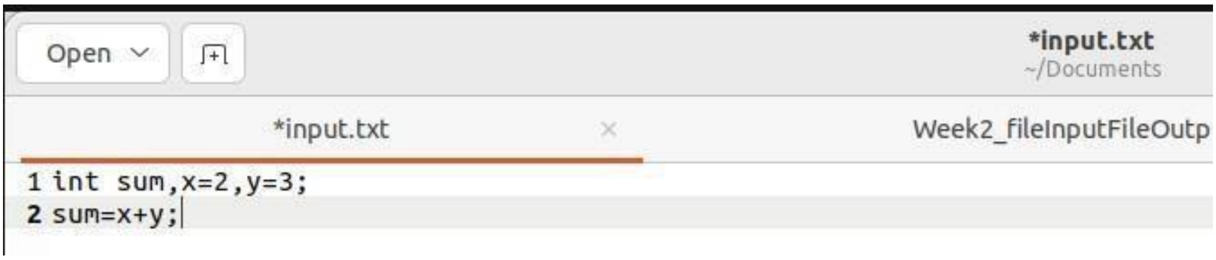
write a lex program to read following input from a file and print valid token on the terminal

```
[0-9]* { printf("%s is Digit \n", yytext); }
[a-zA-Z]* { printf("%s is string \n", yytext);
[a-z,A-Z,0-9]* { printf(" %s is alpha numeric \n",
                 yytext); }

%%
void main()
 {
   printf("Enter the file name!");
   scanf("%s", fname);
   yyin= fopen(fname, "r");
   yylen();
   fclose (yylin);
}
```

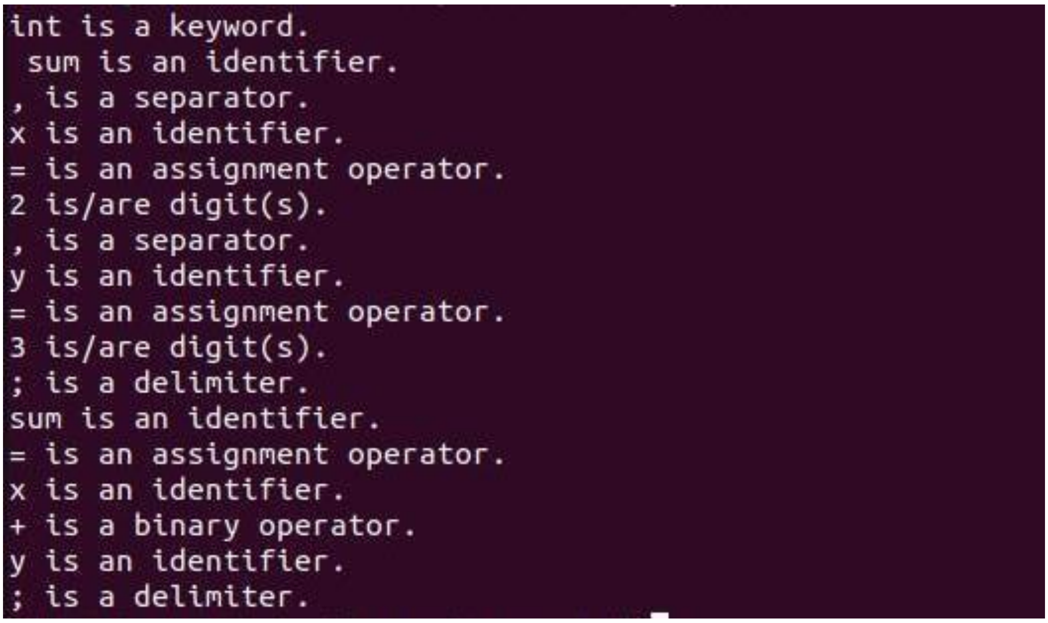
Output:

```
Enter input file name : input-text
123 is Digit
Hello is character.
```

Output



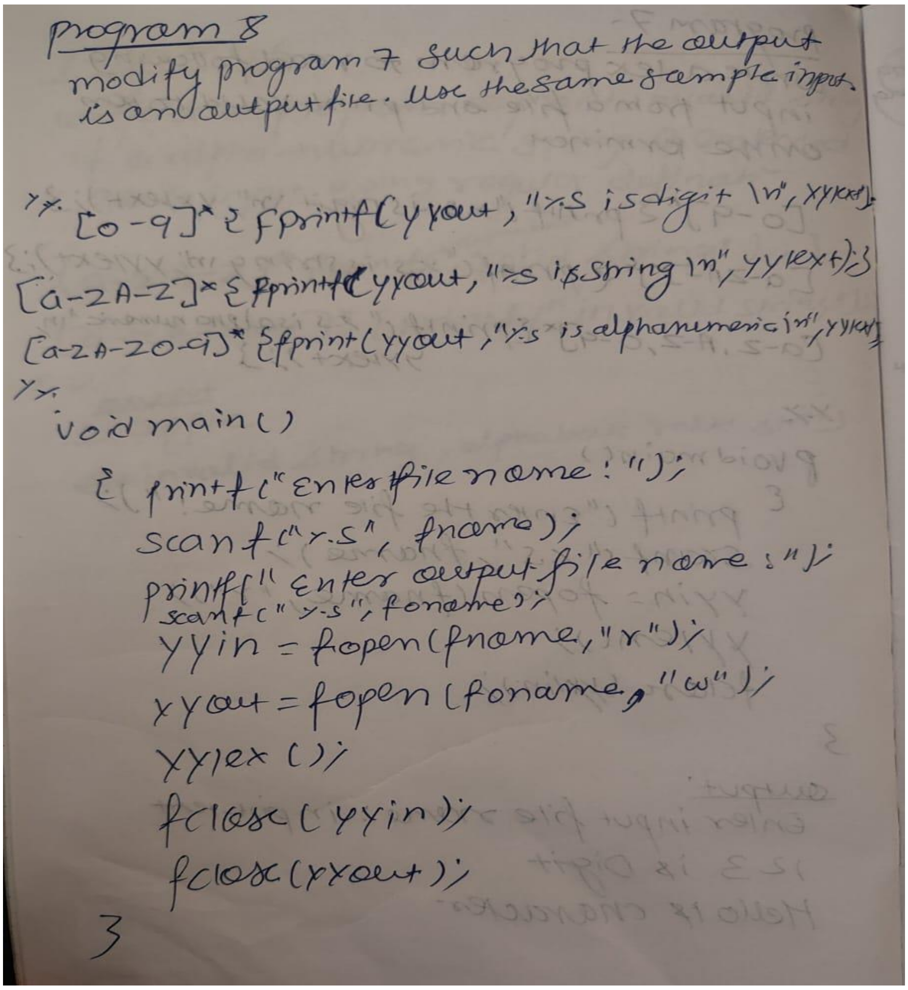
```
int sum,x=2,y=3;
sum=x+y;
```



```
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.

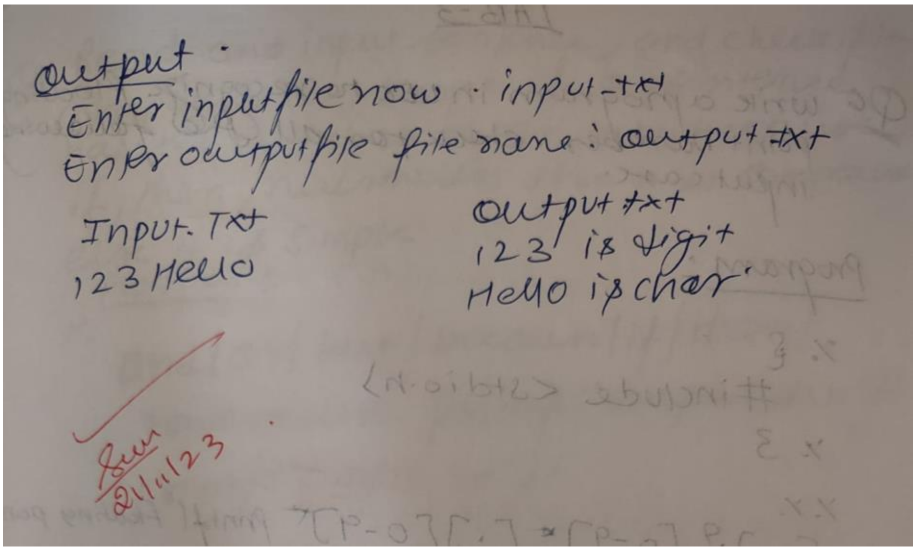
Code



program 8

modify program 7 such that the output is an output file. use the same sample input.

```
%%
[0-9]* {fprintf(yyout,"%s is digit \n",yytext);}
[a-zA-Z]* {fprintf(yyout,"%s is String \n",yytext);}
[a-zA-Z0-9]* {fprint(yyout,"%s is alphanumeric \n",yytext);}
%%
void main()
{ printf("Enter file name: ");
  scanf("%s", fname);
  printf(" enter output file name : ");
  scanf(" %s",foname);
  yyin = fopen(fname,"r");
  yyout=fopen(foname,"w");
  yylex();
  fclose(yyin);
  fclose(yyout);
}
```

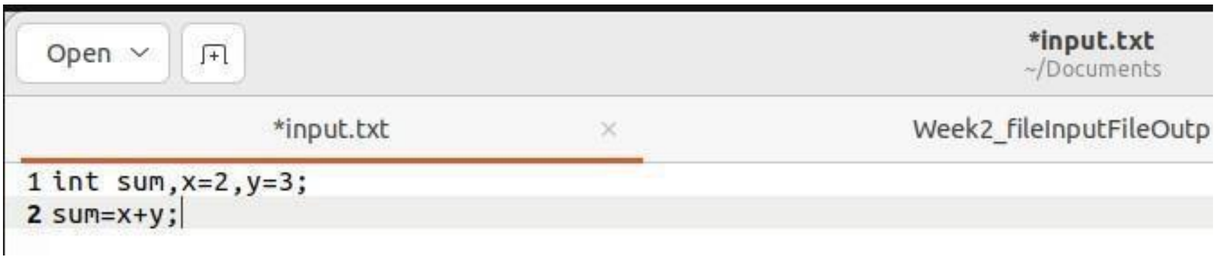


output :

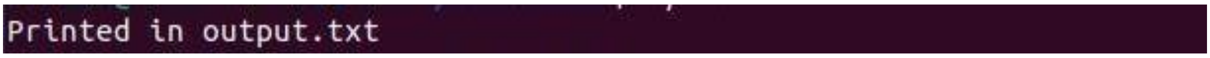
```
Enter inputfile now : input-txt
Enter outputfile file name : output-txt

Input.Txt            output txt
123 Hello            123 is digit
                     Hello is char
```

21/11/23

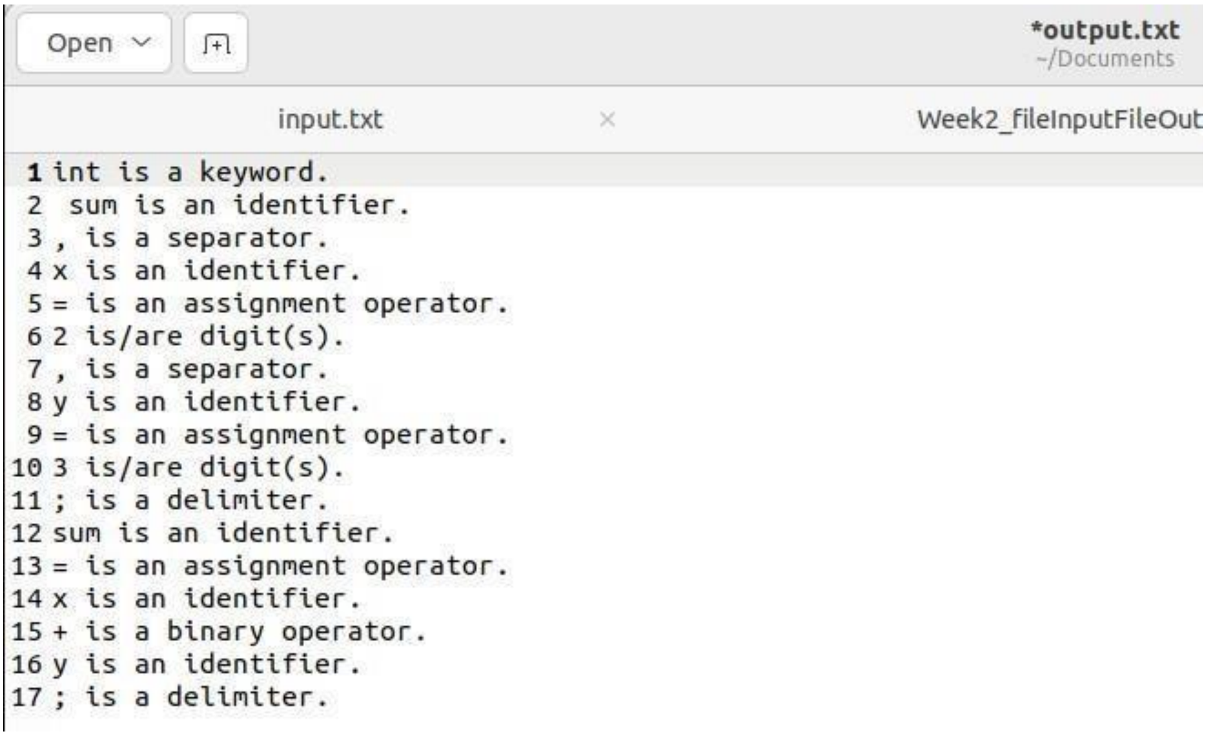
Output



```
int sum,x=2,y=3;
sum=x+y;
```



```
Printed in output.txt
```

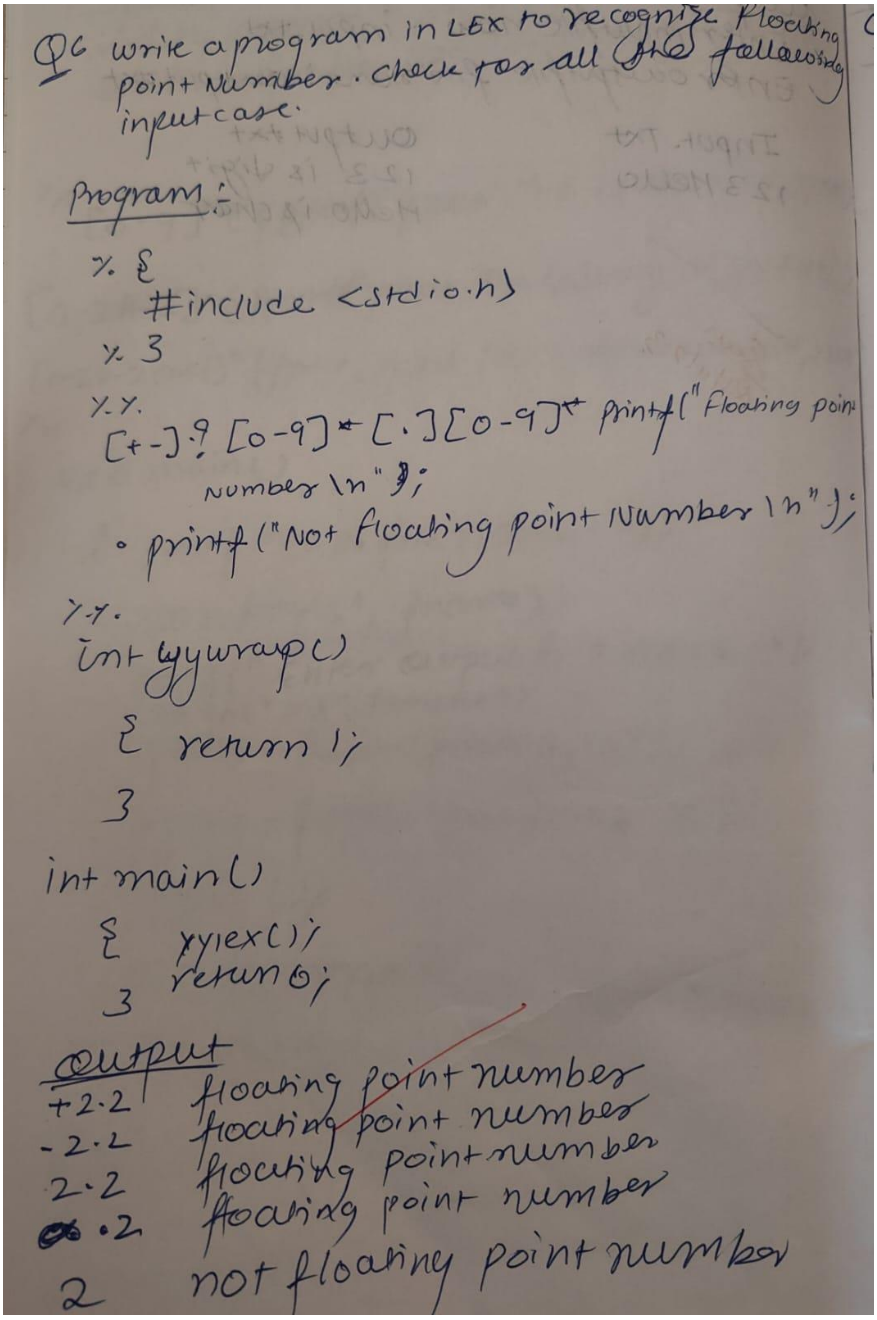


```
int is a keyword.
 sum is an identifier.
, is a separator.
x is an identifier.
= is an assignment operator.
2 is/are digit(s).
, is a separator.
y is an identifier.
= is an assignment operator.
3 is/are digit(s).
; is a delimiter.
sum is an identifier.
= is an assignment operator.
x is an identifier.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

Lab 3

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

Code



Q6 write a program in LEX to recognize floating point Number. Check for all the following inputcase.

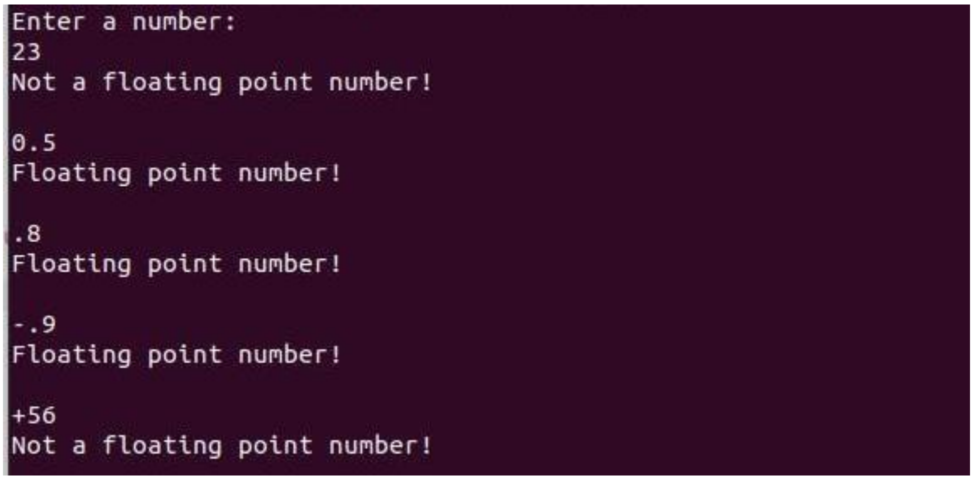
Program :-

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

%%
[+-]? [0-9]* [.][0-9]+ printf("floating point
          number \n");
. printf("Not floating point Number \n");

%%
int yywrap()
  { return 1;
  }

int main()
  { yylex();
    return 0;
  }
```

Output

```
+2.2   floating point number
-2.2   floating point number
2.2    floating point number
.2     floating point number
2      not floating point number
```

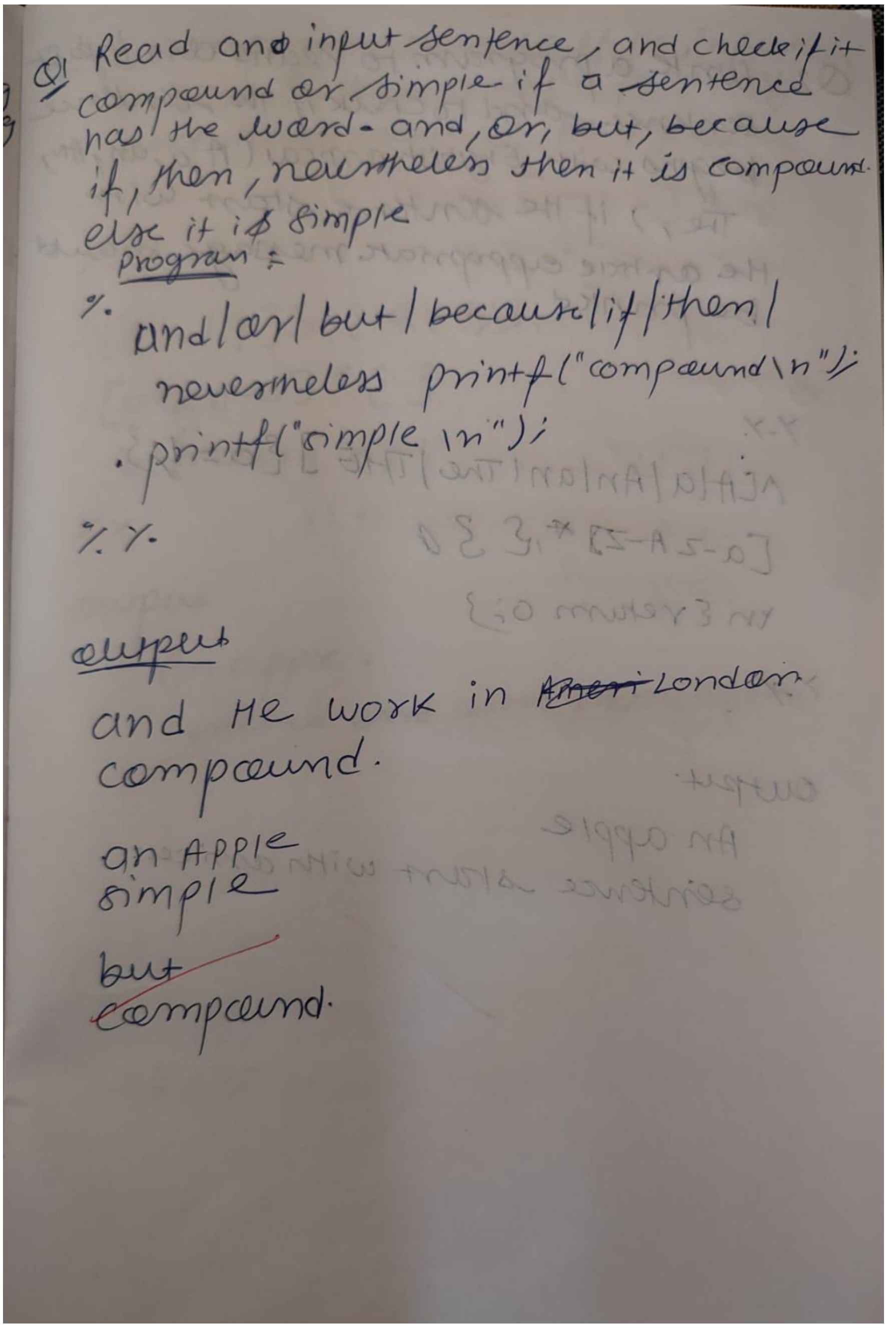
Output



```
Enter a number:
23
Not a floating point number!

0.5
Floating point number!

.8
Floating point number!

-.9
Floating point number!

+56
Not a floating point number!
```

3.2 Read and input sentence, and check if it is compound or simple. If a sentence has the word- and , or ,but ,because ,if ,then ,nevertheless then it is compound else it is simple.

Code



Q! Read and input sentence, and checkifit compound or simple if a ~~sentence~~ has the word- and, or, but, because if, then, neurtheless then it is compound. else it is simple

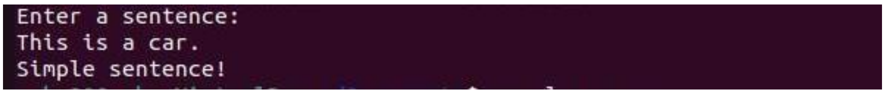
Program :

```
%.
 and/or/ but / because/if/then/
  nevermeless printf("compound\n");
 . printf("simple \n");

%.%.
```

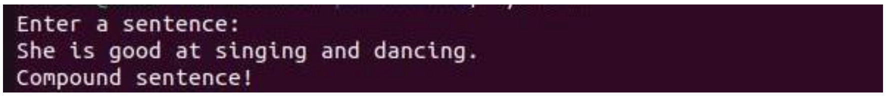
output

```
and He work in ~~Ameri~~ London
compound.

an Apple
simple

but
compound.
```

Output



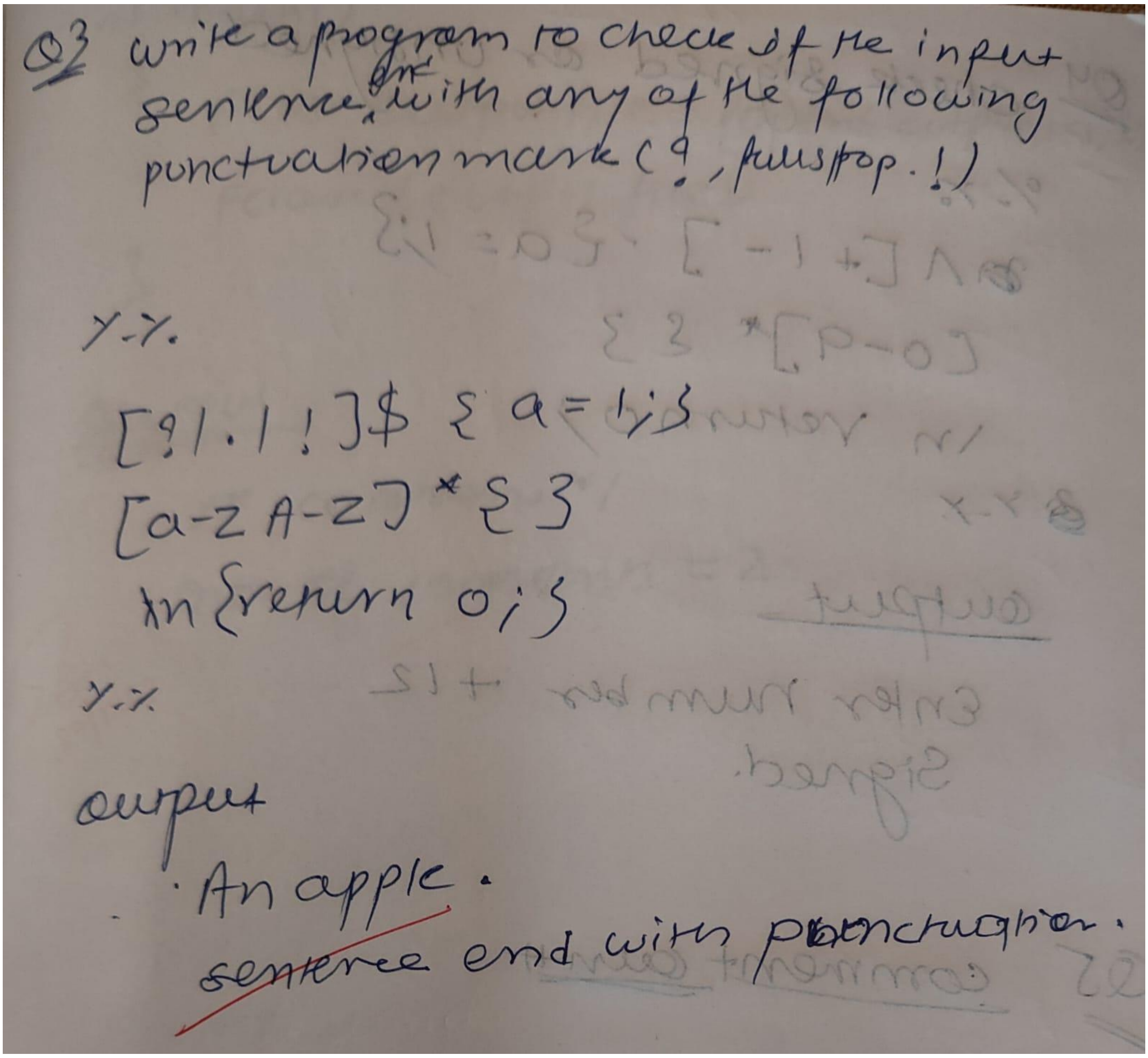
```
Enter a sentence:
This is a car.
Simple sentence!
```



```
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



Q3 write a program to check if the input sentence end with any of the following punctuation mark (?, fullstop . !)

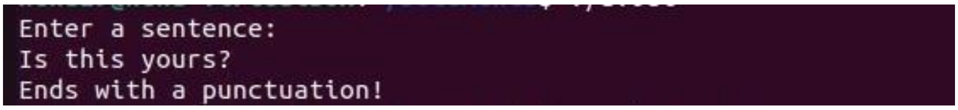
```
%.%

[?|.|!]$ { a=1;}
[a-z A-Z]* { }
\n {return 0;}

%.%
```

output

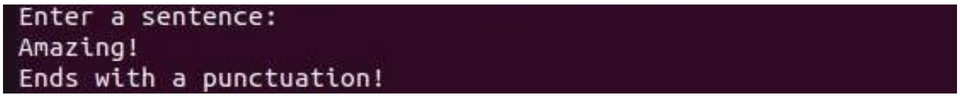
· An apple .

~~sentence~~ end with punctuation .

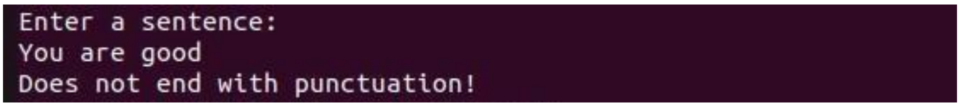
Output



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



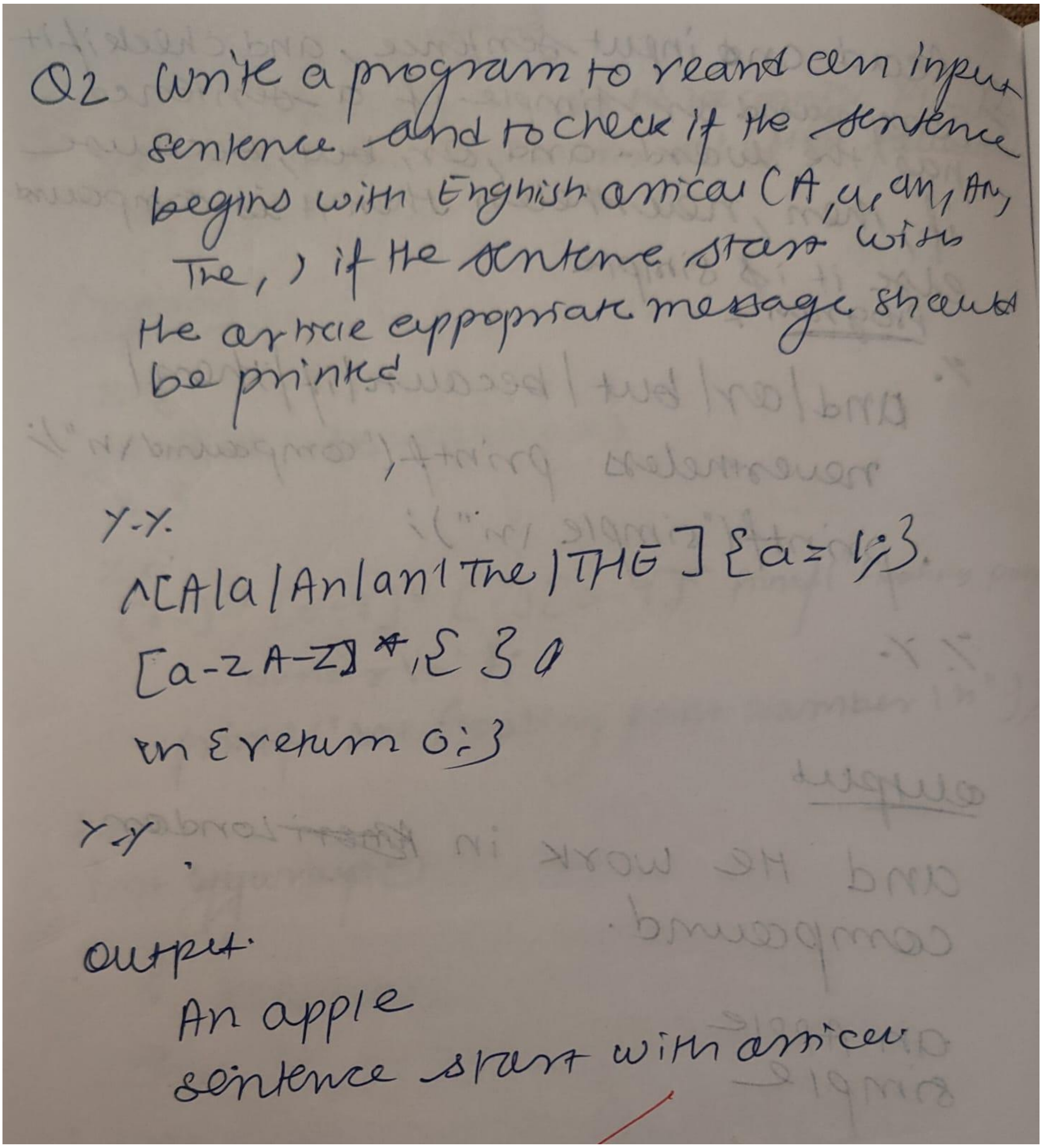
```
Enter a sentence:
Amazing!
Ends with a punctuation!
```



```
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



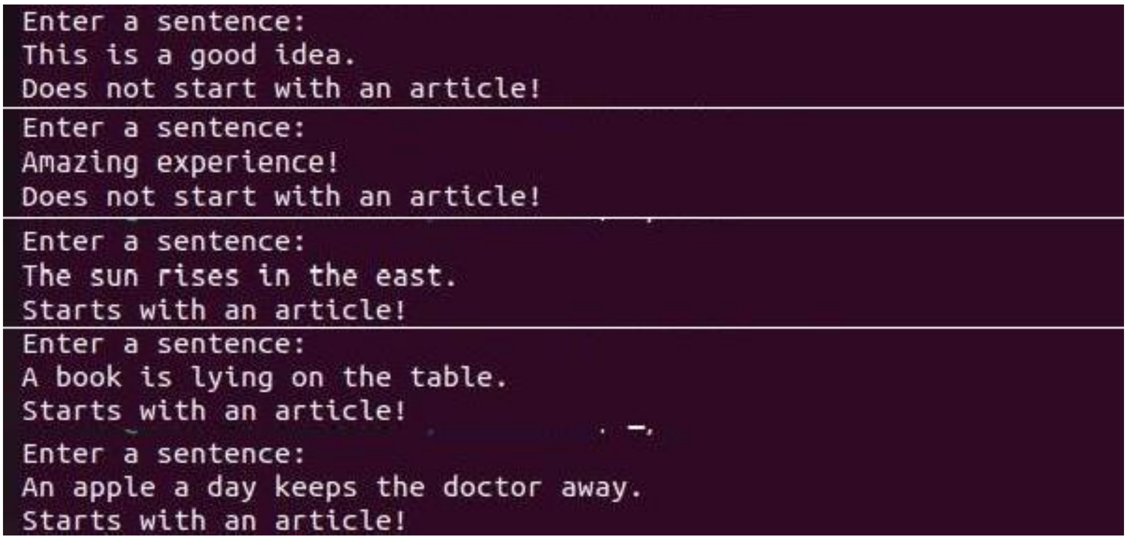
Q2 Write a program to reand an input sentence and to check if the sentence begins with English articel (A, a, an, An, The,) if the sentence start with the article appropriate message should be printed

```
%.%.
^[A|a|An|an|The|THE] {a=1;}
[a-zA-Z]*,{ }
\n {return 0;}
%.%
```

output.

```
An apple
sentence start with articel
```

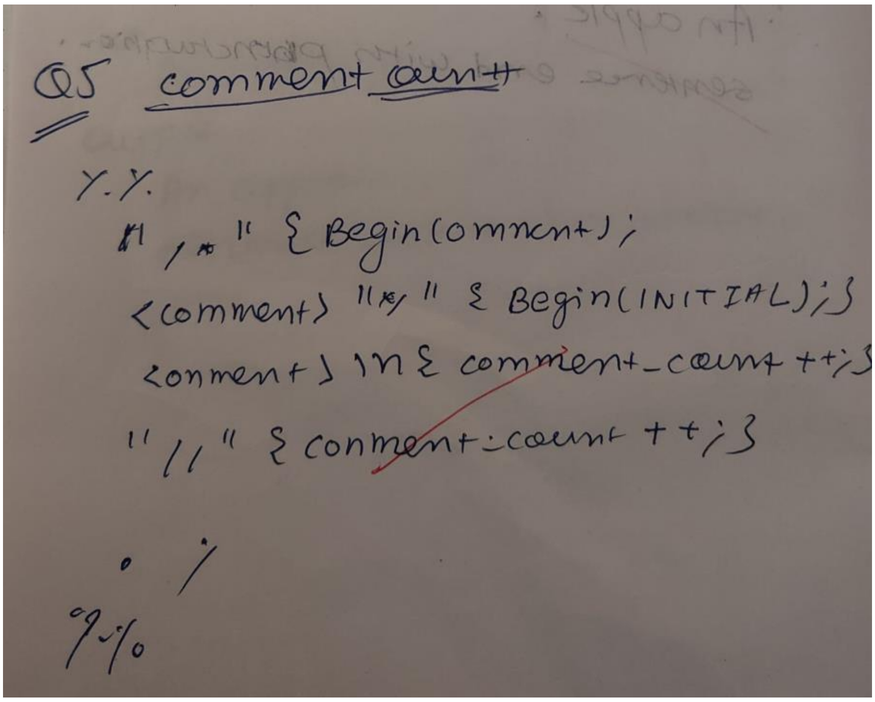
Output



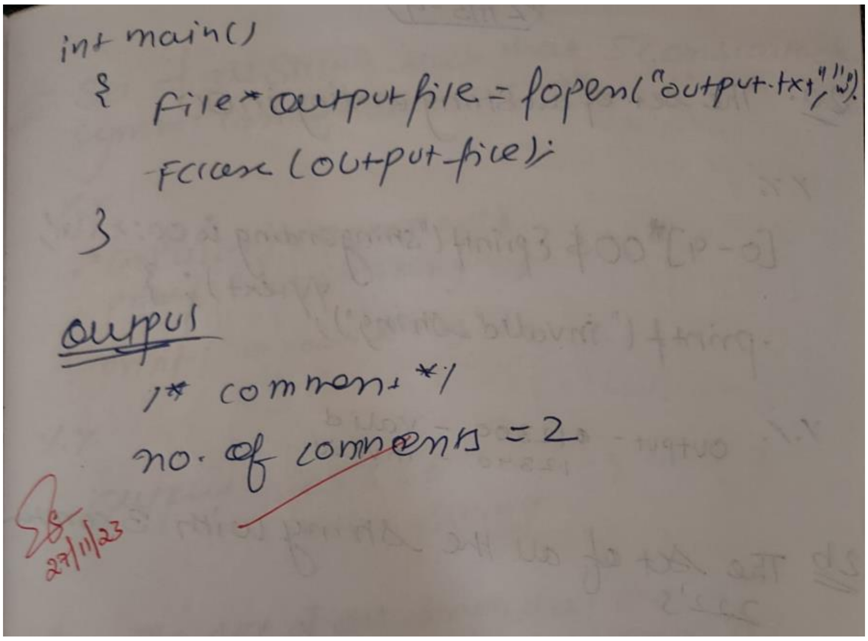
```
Enter a sentence:
This is a good idea.
Does not start with an article!
Enter a sentence:
Amazing experience!
Does not start with an article!
Enter a sentence:
The sun rises in the east.
Starts with an article!
Enter a sentence:
A book is lying on the table.
Starts with an article!
Enter a sentence:
An apple a day keeps the doctor away.
Starts with an article!
```

3.5 Lex program to count the number of comment lines (multi line comments or single line) in a program. Read the input from a file called input.txt and print the count in a file called output.txt.

Code



```
Q5 comment count++

%%
"/*" { Begin(omment);
<comment> "*/" { Begin(INITIAL);}
<onment> \n { comment_count ++;}
"//" { conment_count ++;}

. ;
%%
```



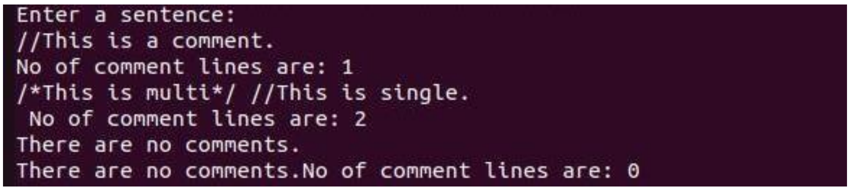
```
int main()
{ File*outputfile = fopen("output.txt","w");
  Fclose (output_file);
}
```

ouput

```
/* comment */
no. of comments = 2
```

27/11/23

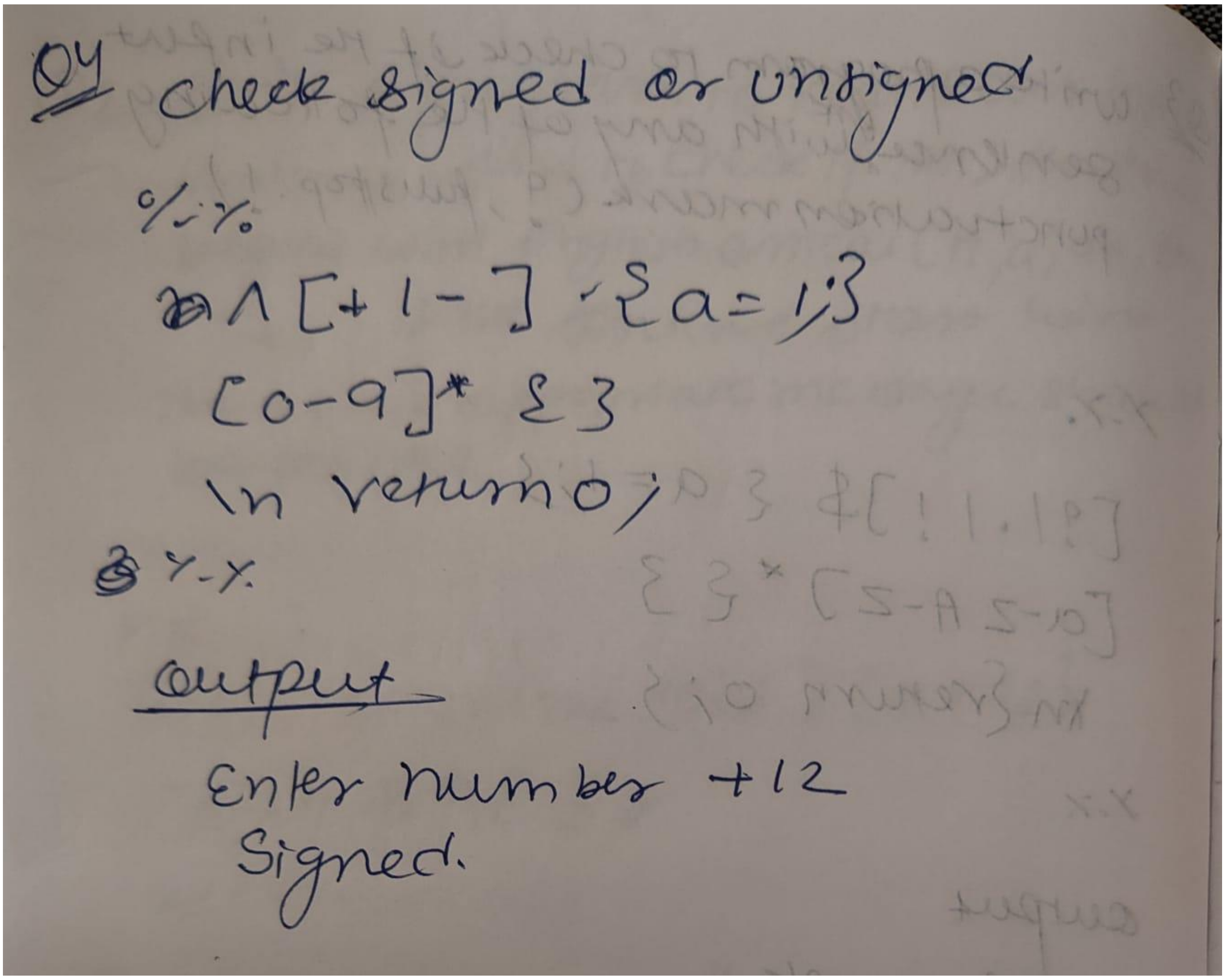
Output



```
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



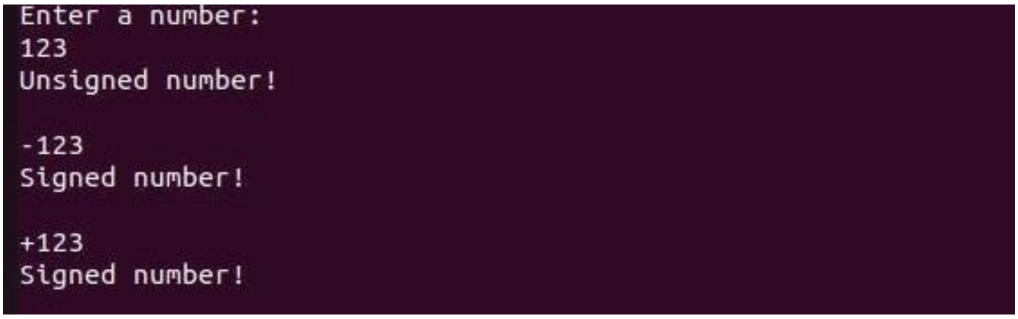
Q4 check signed or unsigned

```
%-%
^[+|-]·{a=1;}
 [0-9]* {}
 \n return 0;
%-%.
```

output

Enter number +12
Signed.

Output

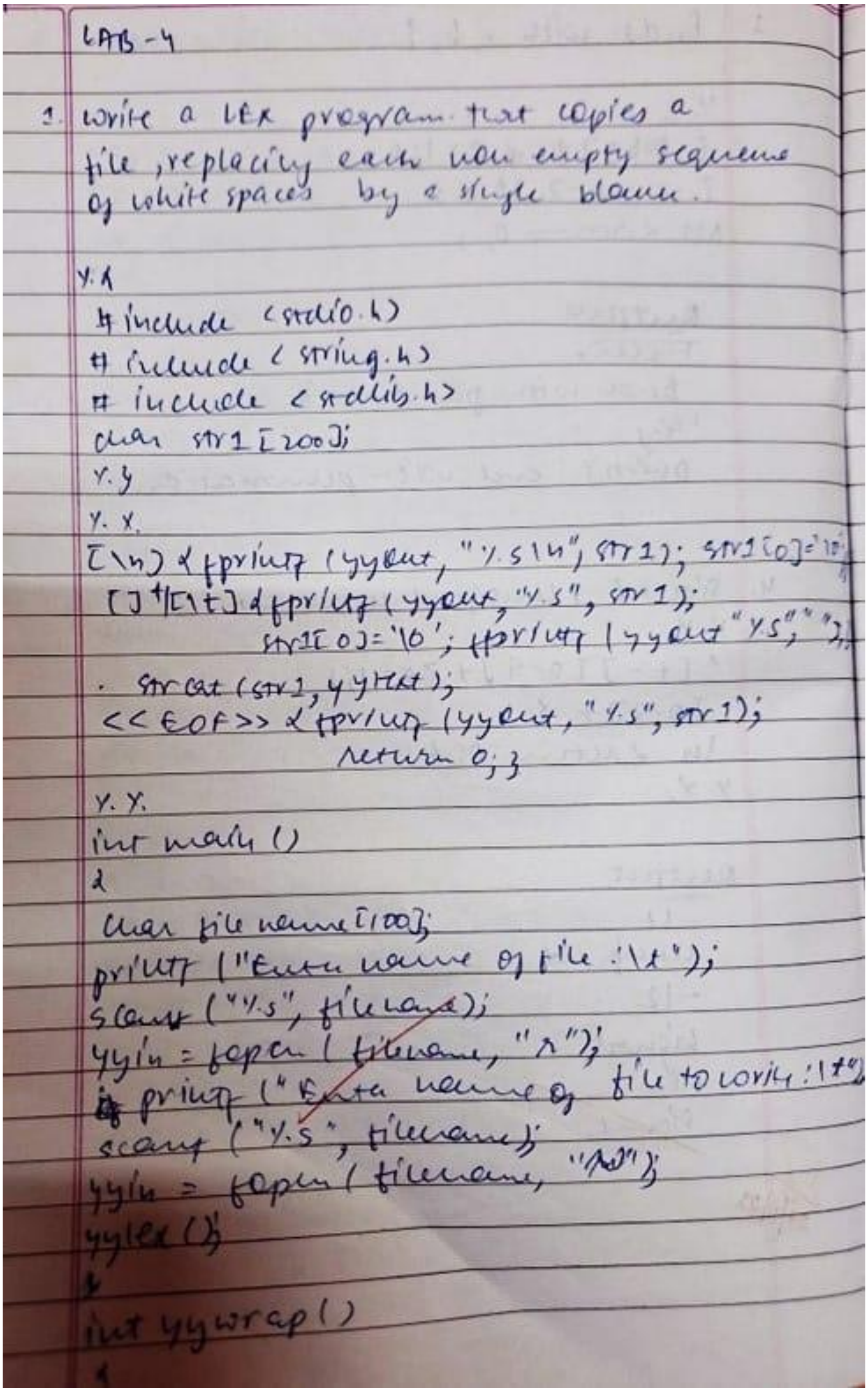


```
Enter a number:
123
Unsigned number!

-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

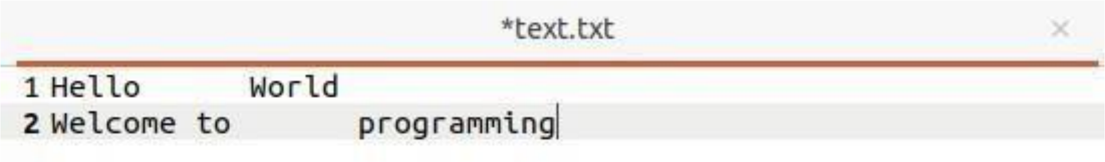


LAB - 4

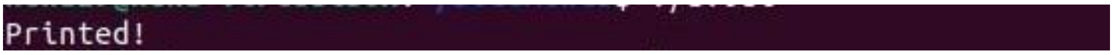
1. write a LEX program that copies a file, replacing each non empty sequence of white spaces by a single blank.

```
%{
 #include <stdio.h>
 # include < string.h>
 # include < stdlib.h>
 char str1 [200];
%}
%%
[\n] {fprintf (yyout, "%s\n", str1); str1[0]='\0';
 [ ]+|[\t] {fprintf(yyout, "%s", str1);
              str1[0]='\0'; fprintf (yyout "%s", " ");
 .  strcat (str1, yytext);
 <<EOF>> {fprintf (yyout, "%s", str1);
                 return 0;}
%%
int main ()
{
 char file name[100];
 printf ("Enter name of file :\t");
 scanf ("%s", filename);
 yyin = fopen ( filename, "r");
 printf ("Enter name of file to write :\t");
 scanf ("%s", filename);
 yyin = fopen ( filename, "w");
 yylex ();
}
int yywrap ()
{
```

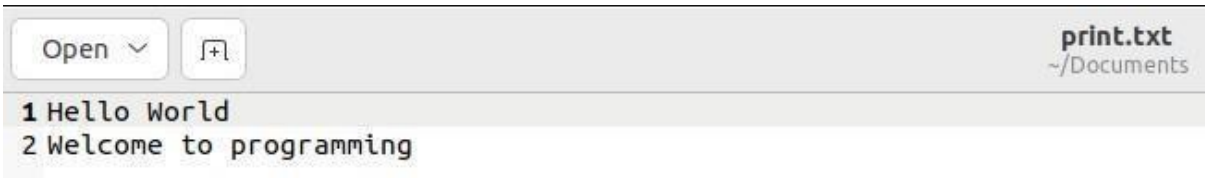
Output



```
Hello       World
Welcome to          programming
```



```
Printed!
```

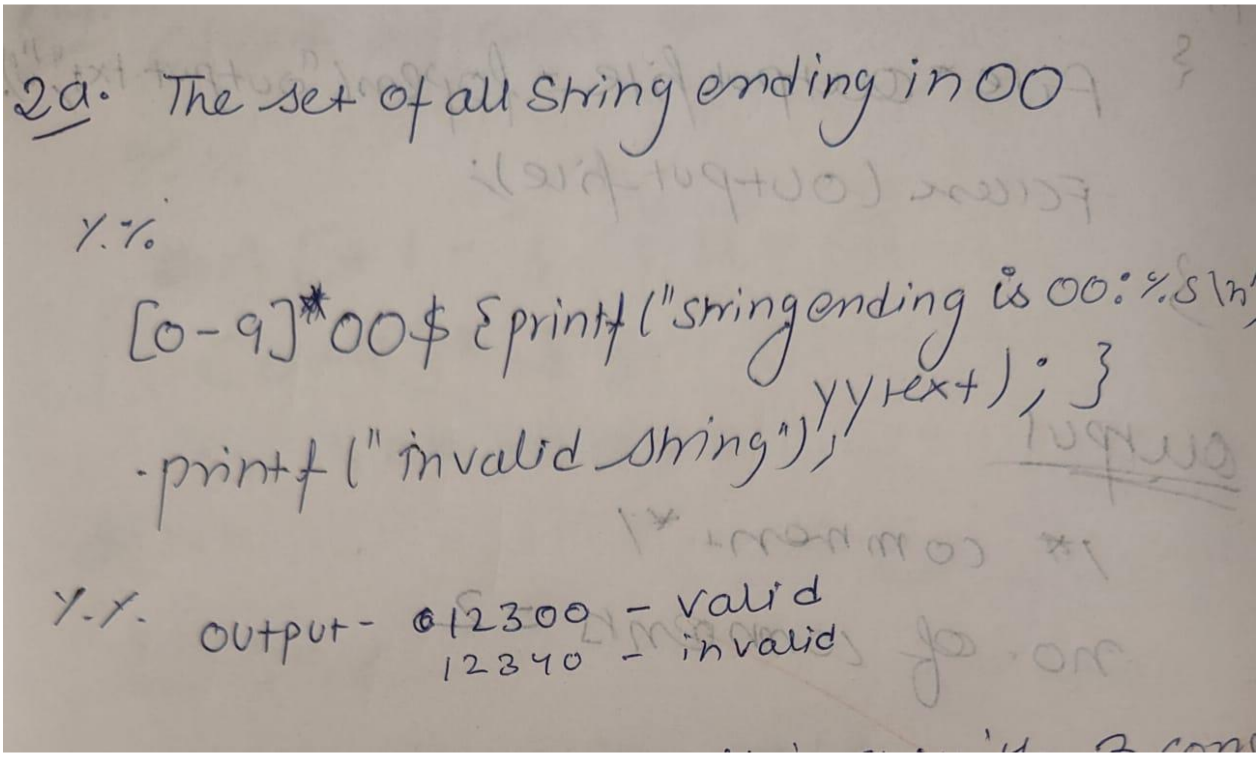


```
Hello World
Welcome to programming
```

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.

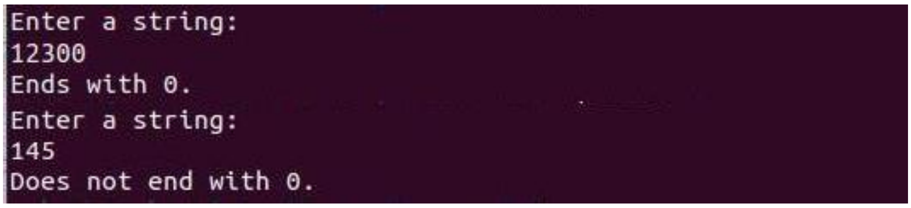
Code



2a. The set of all String ending in 00

```
%%
[0-9]*00$ {printf("stringending is 00:%s\n",
                 yytext); }
.printf("invalid string");

%%   output-  12300 - valid
              12340 - invalid
```

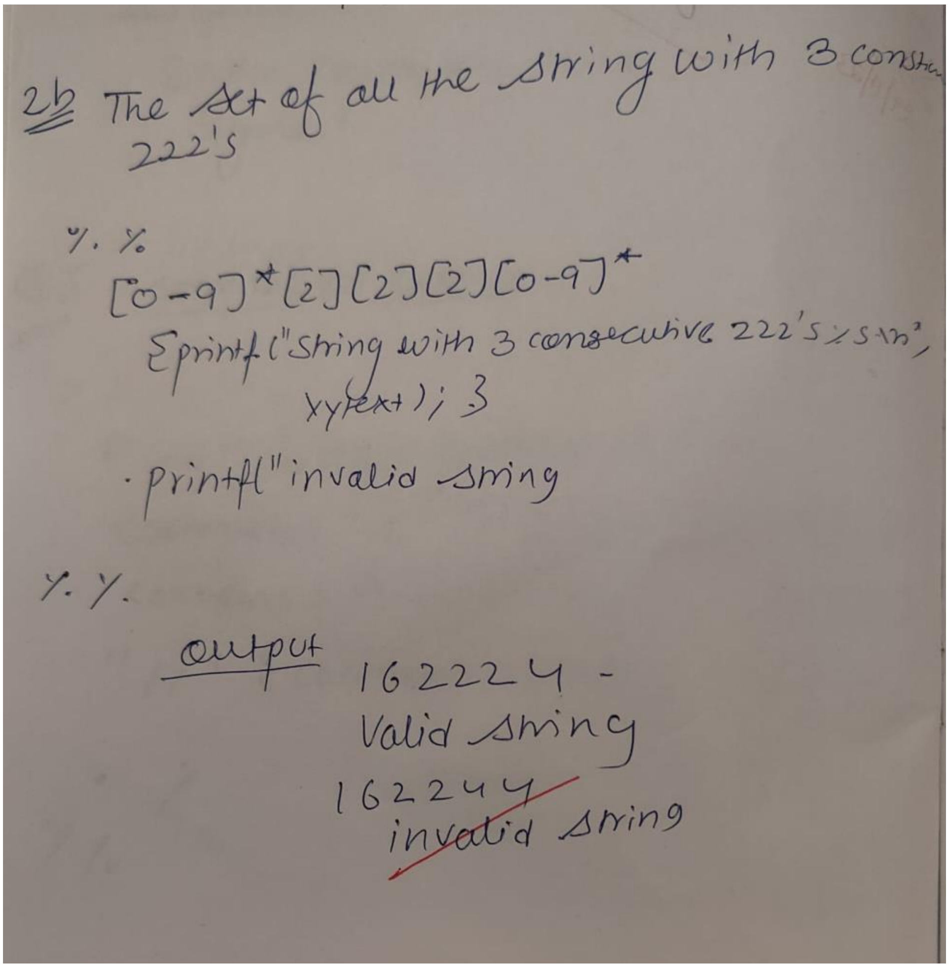
Output



```
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

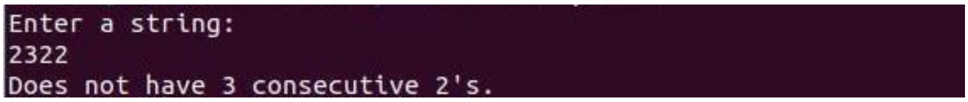


2b The set of all the string with 3 consta
222's

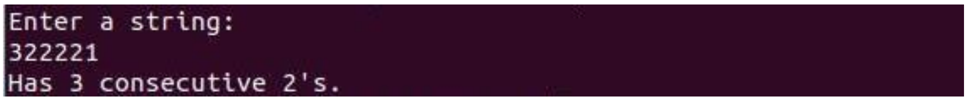
```
%.%
[0-9]*[2][2][2][0-9]*
 {printf("string with 3 consecutive 222's : %s\n",
          yytext); }
.printf("invalid string
%.%.
```

output 162224 -
Valid string
162244
invalid string

Output



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

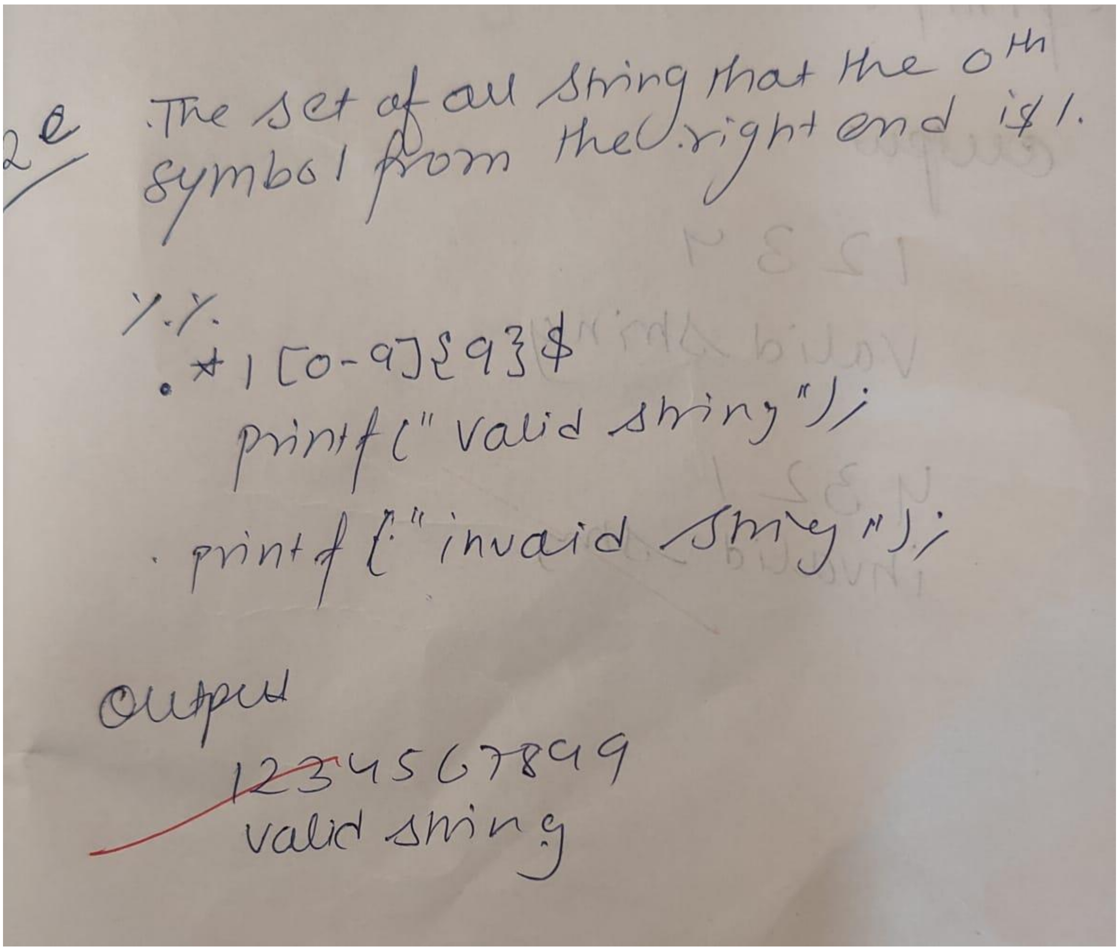


```
Enter a string:
322221
Has 3 consecutive 2's.
```

28

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

Code



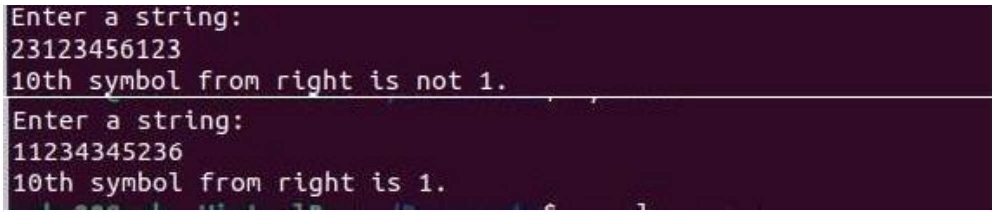
2e The set of all string that the 0th symbol from the right end is 1.

```
%%
.*1[0-9]{9}$
    printf("valid string");
 printf("invaid string");
```

Output

1234567899
valid string

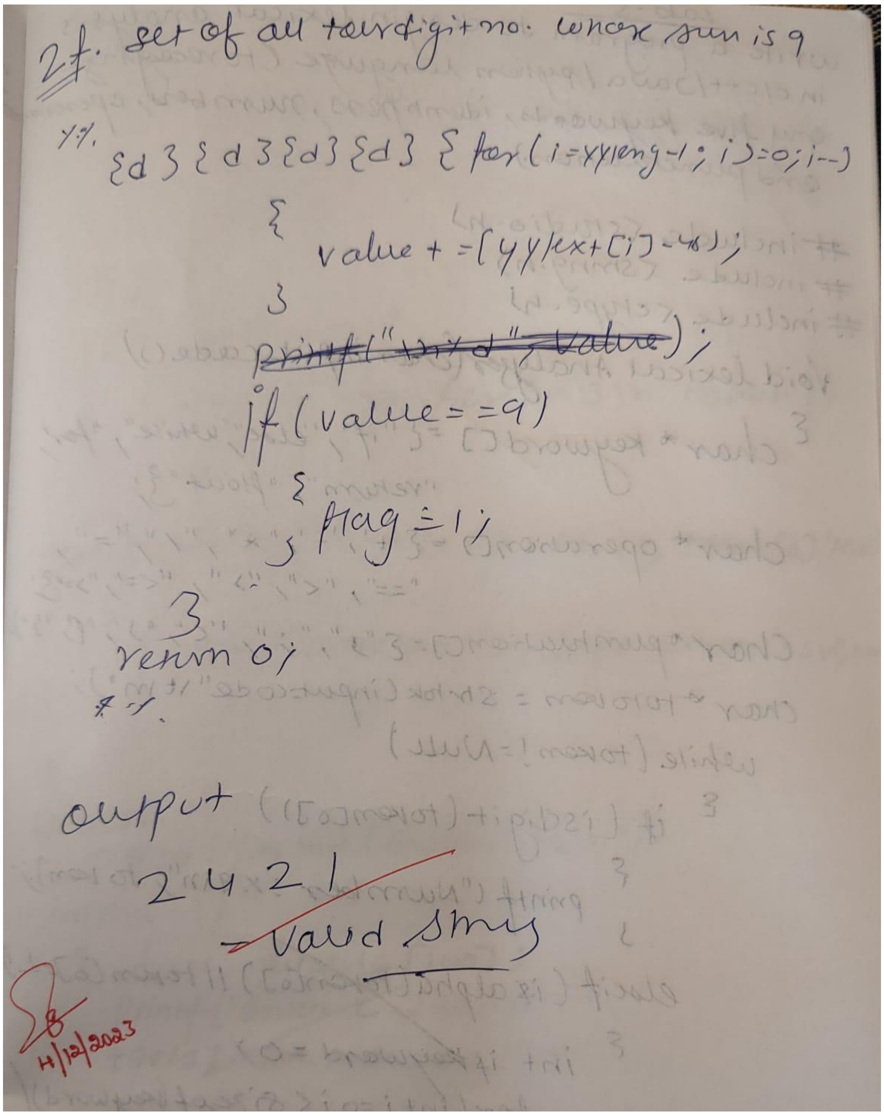
Output



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

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

Code

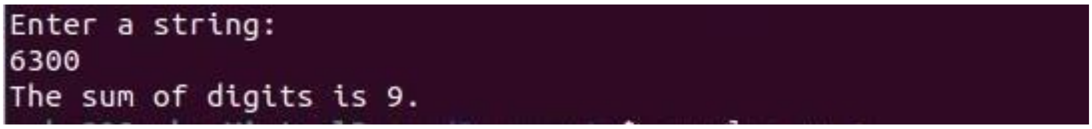


```
2f. set of all fourdigit no. whose sum is 9

%%
{d}{d}{d}{d} { for(i=yyleng-1; i>=0;i--)
        {
            value + =(yytext[i]-48);
        }
        ~~printf("\n%d", value~~);
        if(value= =9)
          {
            flag =1;
          }
  }
return 0;
%.

output

    2421
     - Valid string
```

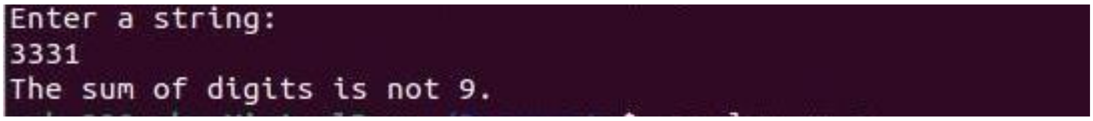
4/12/2023

4.2.5

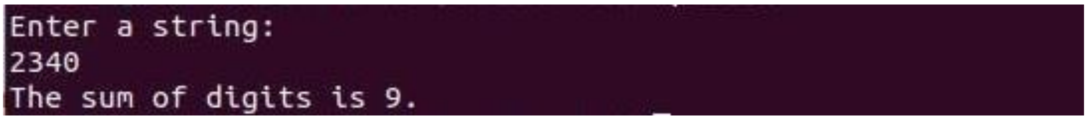
Output



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



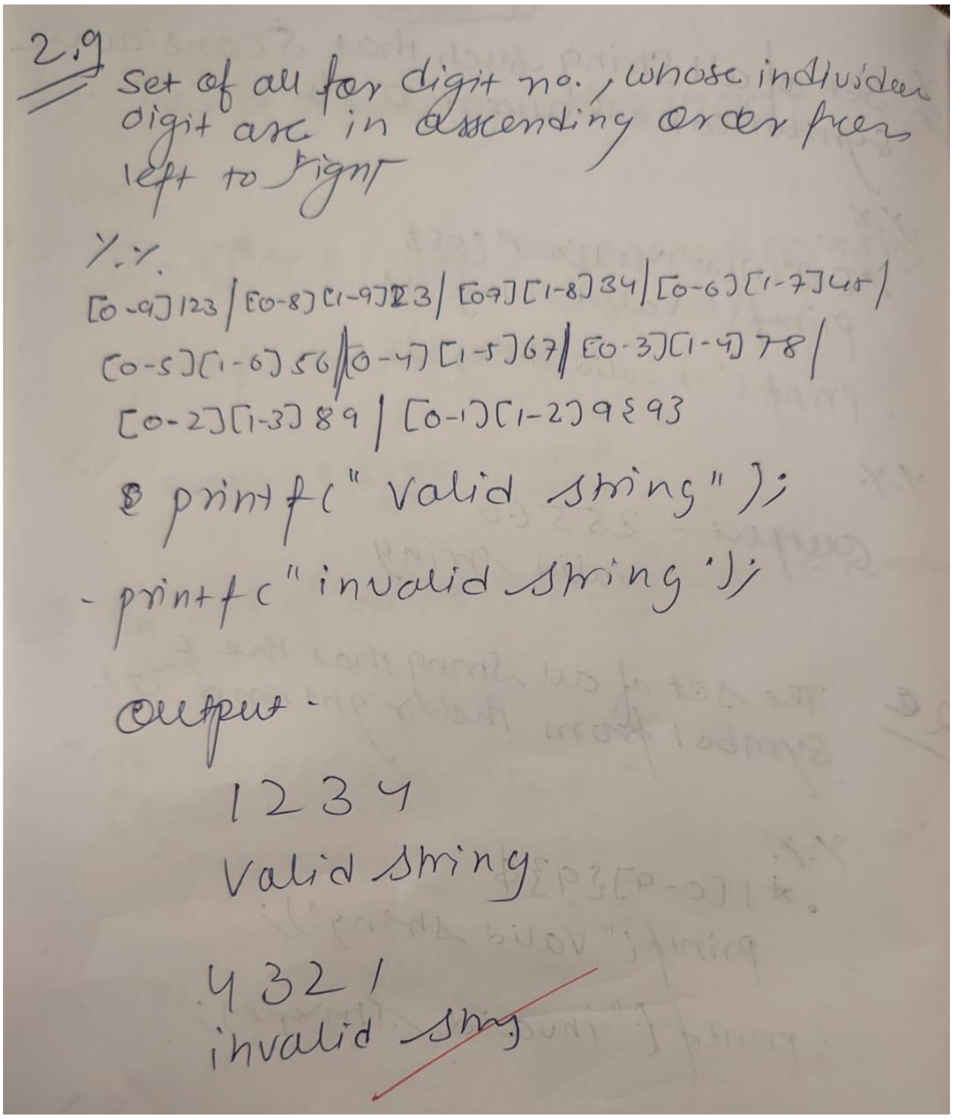
```
Enter a string:
3331
The sum of digits is not 9.
```



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

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

Code



2.9 Set of all for digit no., whose individual digit are in asscending order from left to right

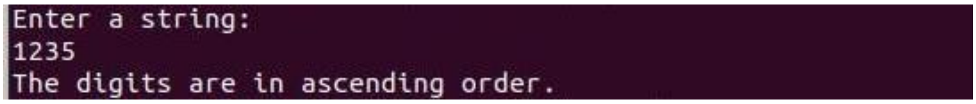
```
%%
[0-9]123|[0-8][1-9]23|[0-7][1-8]34|[0-6][1-7]45|
[0-5][1-6]56|[0-4][1-5]67|[0-3][1-4]78|
[0-2][1-3]89|[0-1][1-2]9{9}
printf("valid string");
- printf("invalid string");
```

Output -

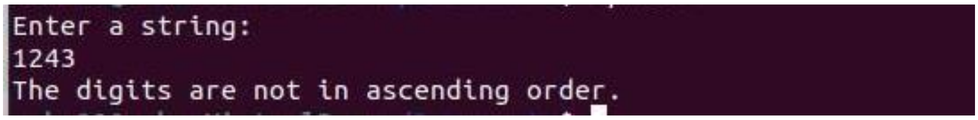
1234
Valid string

4321
invalid string

Output



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

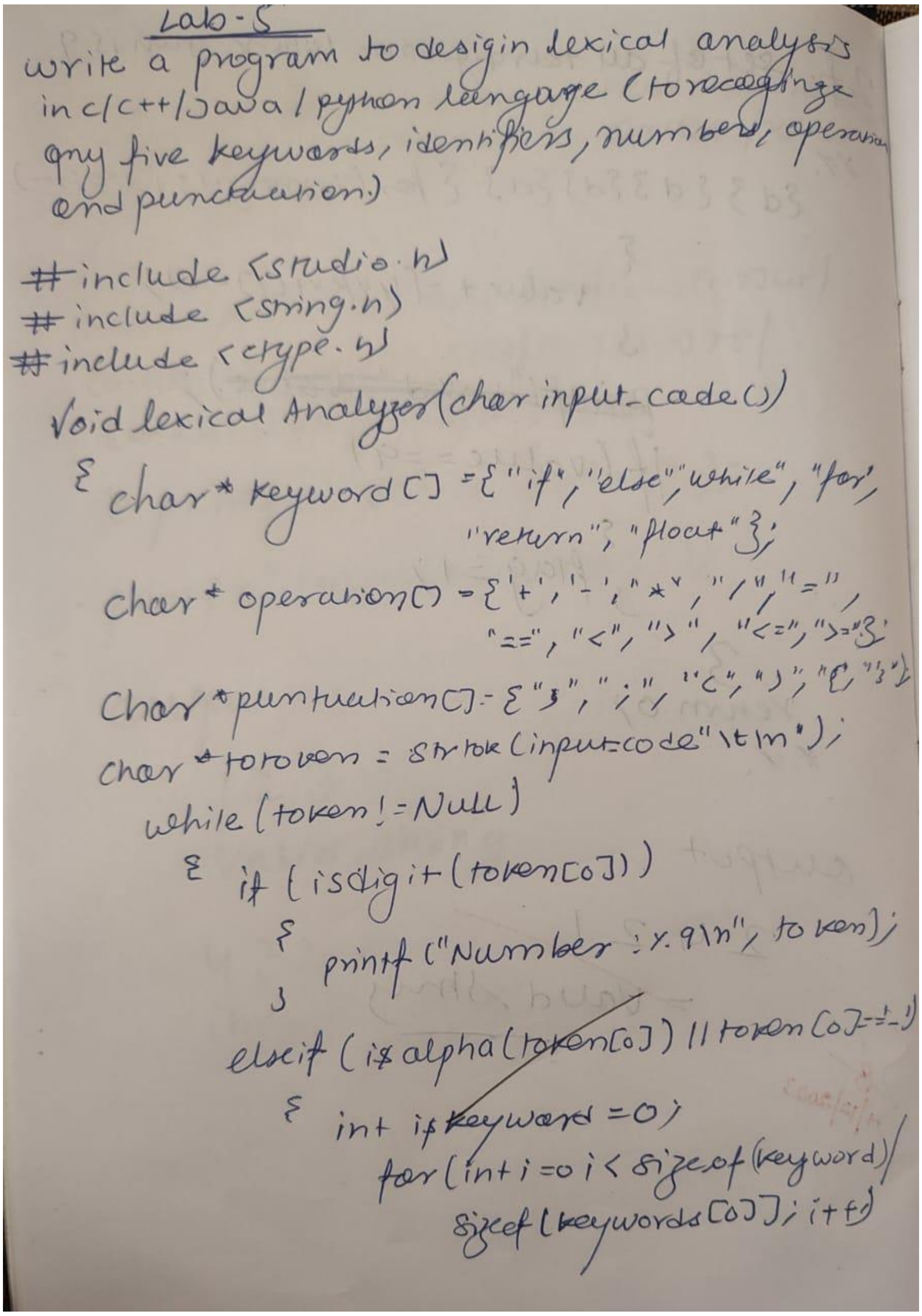


```
Enter a string:
1243
The digits are not in ascending order.
```

Lab 5

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

Code

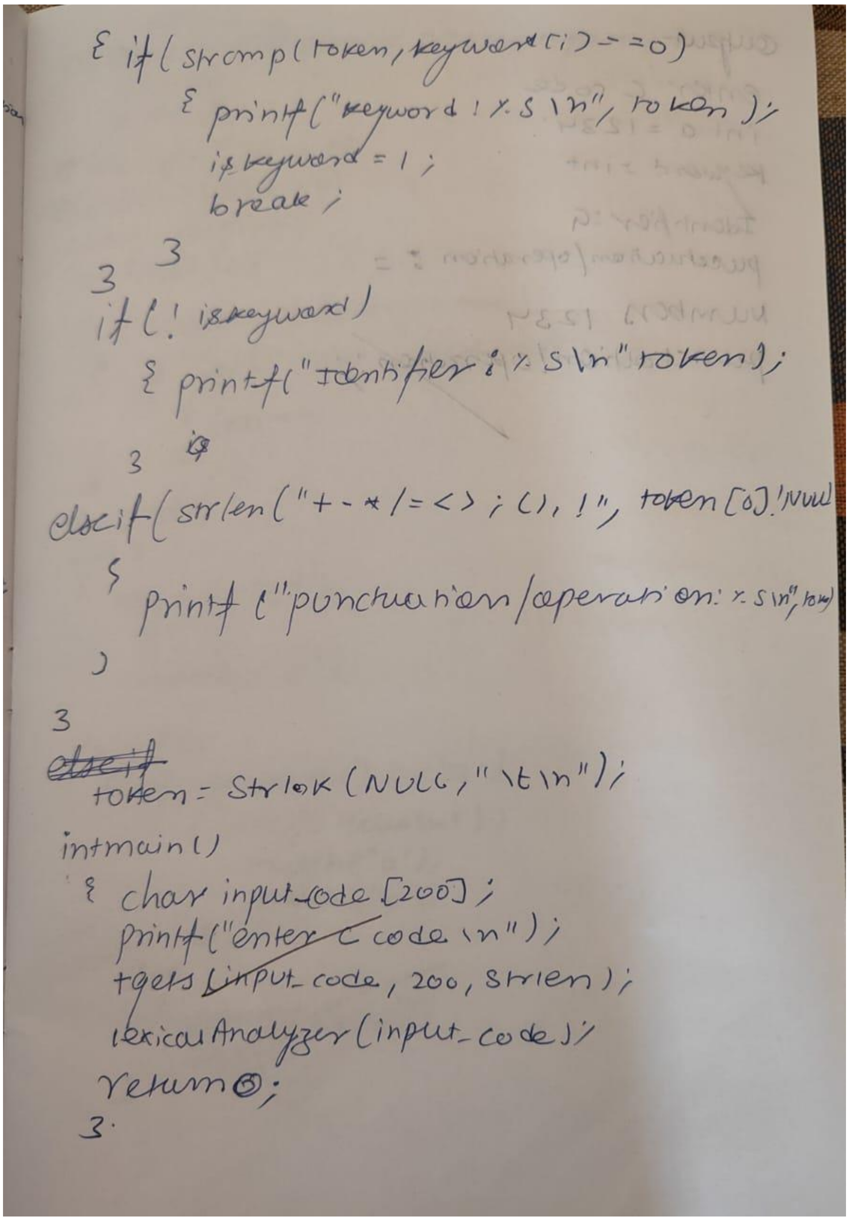


Lab-5

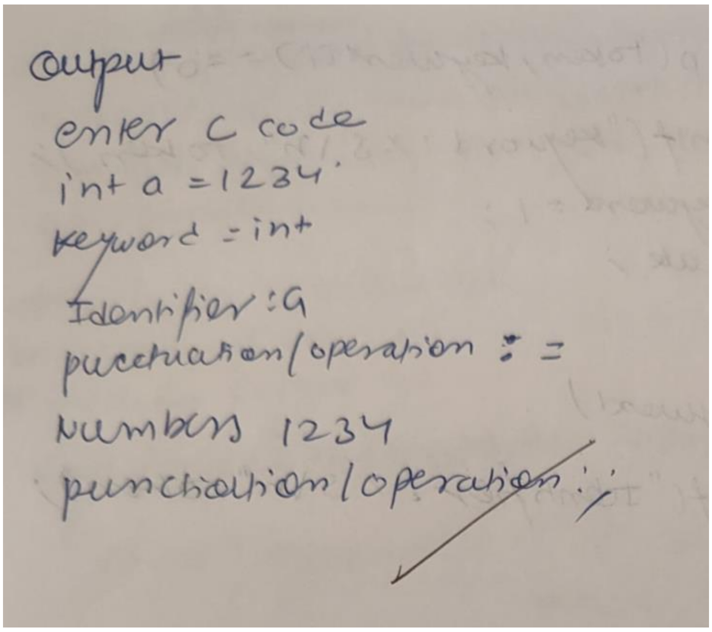
write a program to desigin lexical analysis in c/c++/Java/python language (to recogingze any five keywords, identifiers, numbers, operation and punctuation)

```
#include <studio.h>
#include <string.h>
#include <ctype.h>
Void lexical Analyzer(char input_code())
 { char* keyword[] ={"if","else","while","for",
                     "return", "float"};
   char* operation() ={'+','-',"*","/","=",
                     "==","<",">","<=",">="};
   Char*puntuation[]-{"}",";","(",")","{","}"}
   char* torovenn = strtok(input=code"\t\n");
     while(token!=Null)
       { if (isdigit(token[0]))
           {
             printf("Number :%q\n", token);
           }
         elseif (isalpha(token[0]) || token[0]=='_')
           { int iskeyword =0;
               for(int i=0 i< sizeof(keyword)/
                   sizeof(keywords[0]); i++)
```

<div style="text-align:center"> </div>

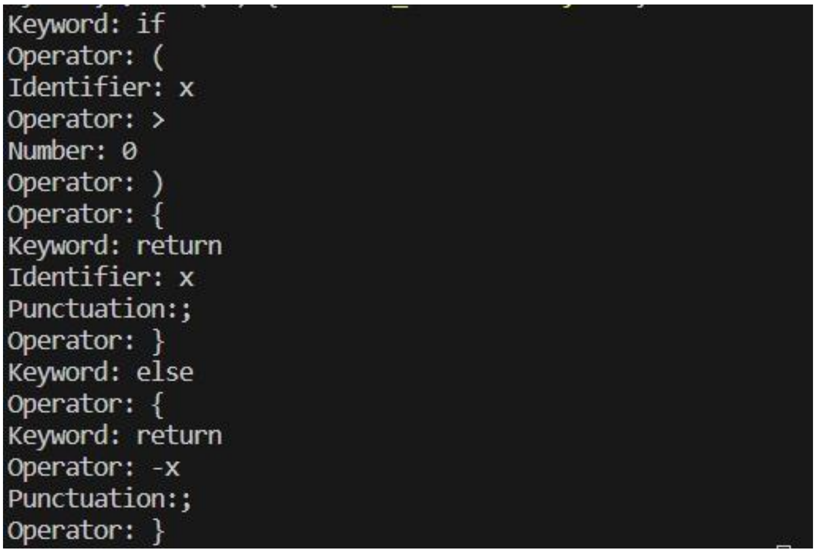


```
{ if(strcmp(token,keyword[i])==0)
      { printf("keyword : %s \n", token );
        iskeyword = 1 ;
        break ;
      }
  }
  if(! iskeyword)
    { printf("Identifier : %s\n" token);
       is
    }
elseif(strlen("+-*/=<>;(),!", token[0]!NULL
  {
     printf ("punctuation/operation: %s\n", token)
  )
}
~~elseif~~
  token = Strtok(NULL, " \t\n");
intmain()
  { char input-code [200];
    printf("enter C code \n");
    fgets (input_code, 200, strlen);
    lexicalAnalyzer(input_code);
    return0;
  }.
```



```
output
enter C code
int a =1234.
keyword = int
Identifier :a
puctuation/operation : =
Numbers 1234
punctiation/operation : ;
```

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: }
```

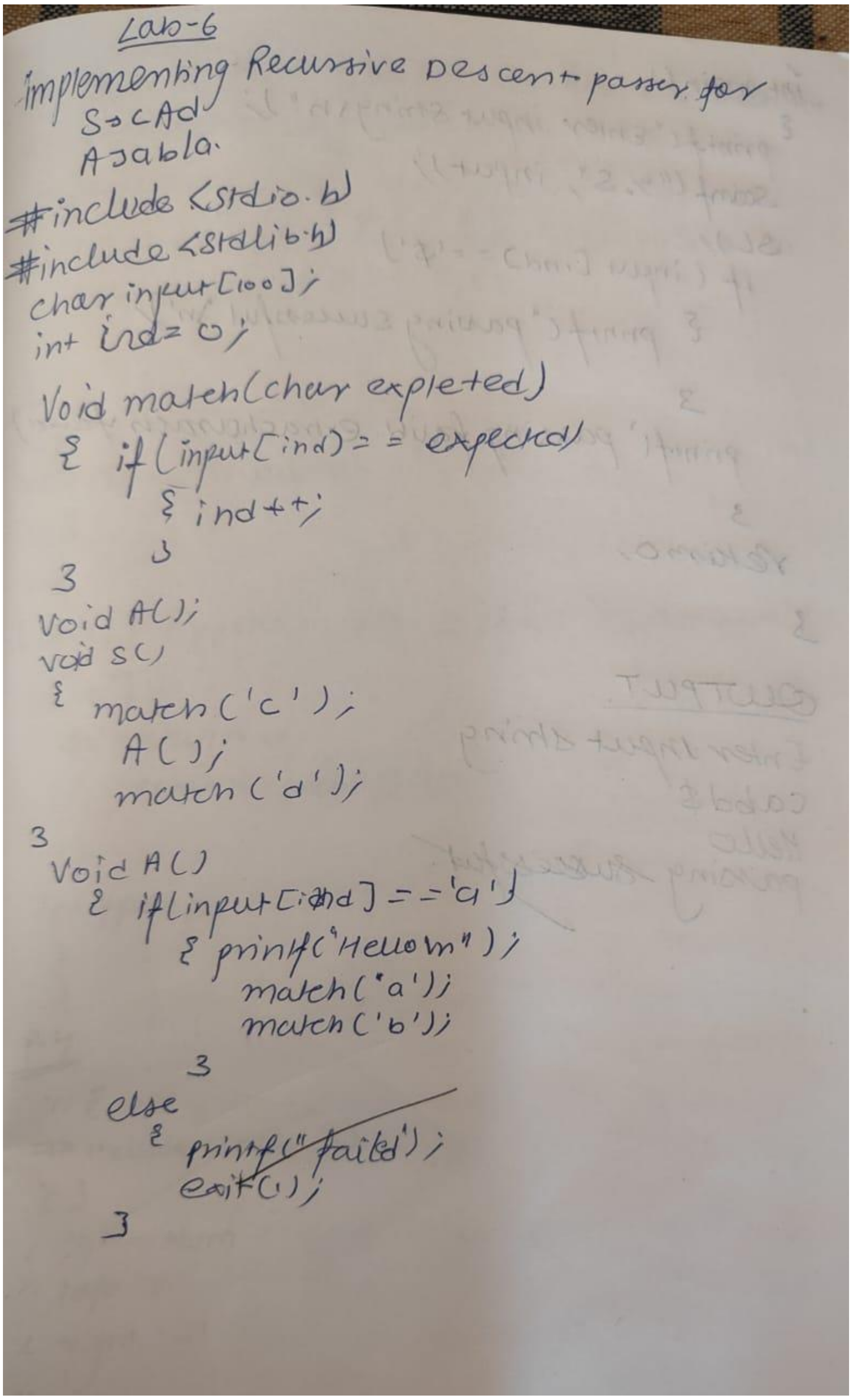
Lab 6

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

S->cAd

A->ab | a

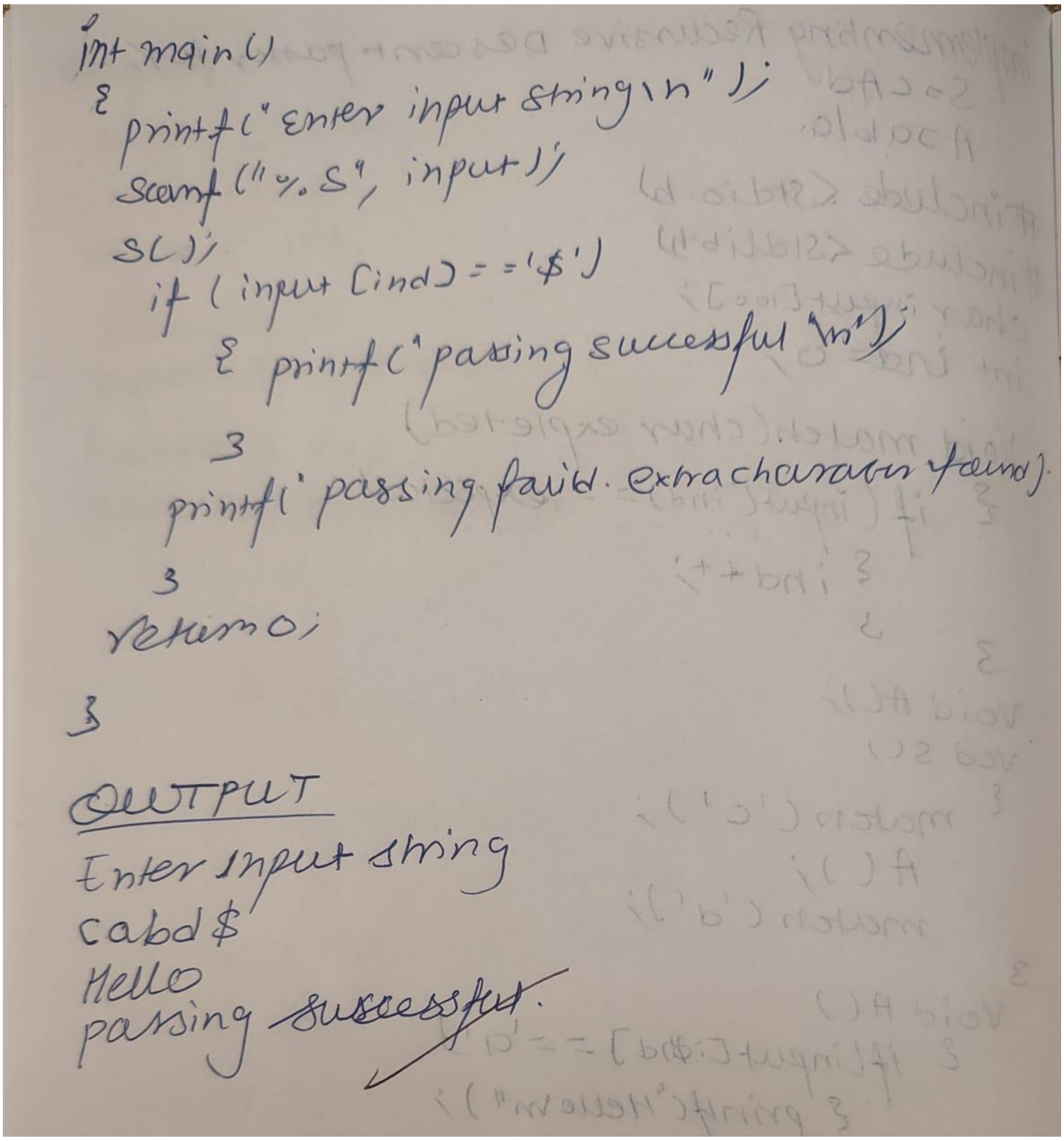
Code



Lab-6

implementing Recursive Descent passer for
S->cAd
A->ab|a.

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

Void match(char expleted)
 { if(input[ind]== expectd)
     { 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);
  }
```

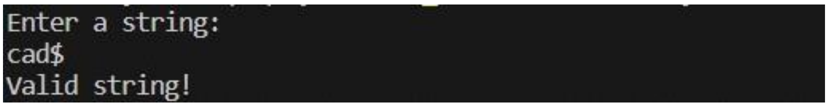


```
int main()
 {
  printf("Enter input string\n");
  scanf("%s", input);
  S();
   if (input [ind] = = '$')
       { printf("passing successful \n");
       }
    printf(" passing faild. extracharacter found).
    }
  return 0;
}
```

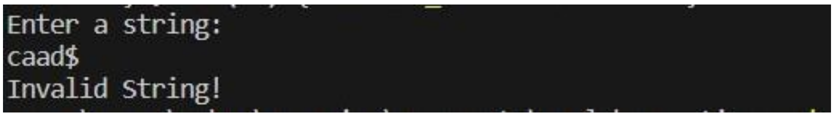
OUTPUT

```
Enter input string
cabd$
Hello
passing successful.
```

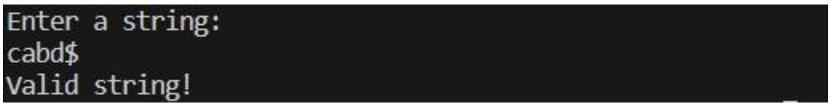
Output



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



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

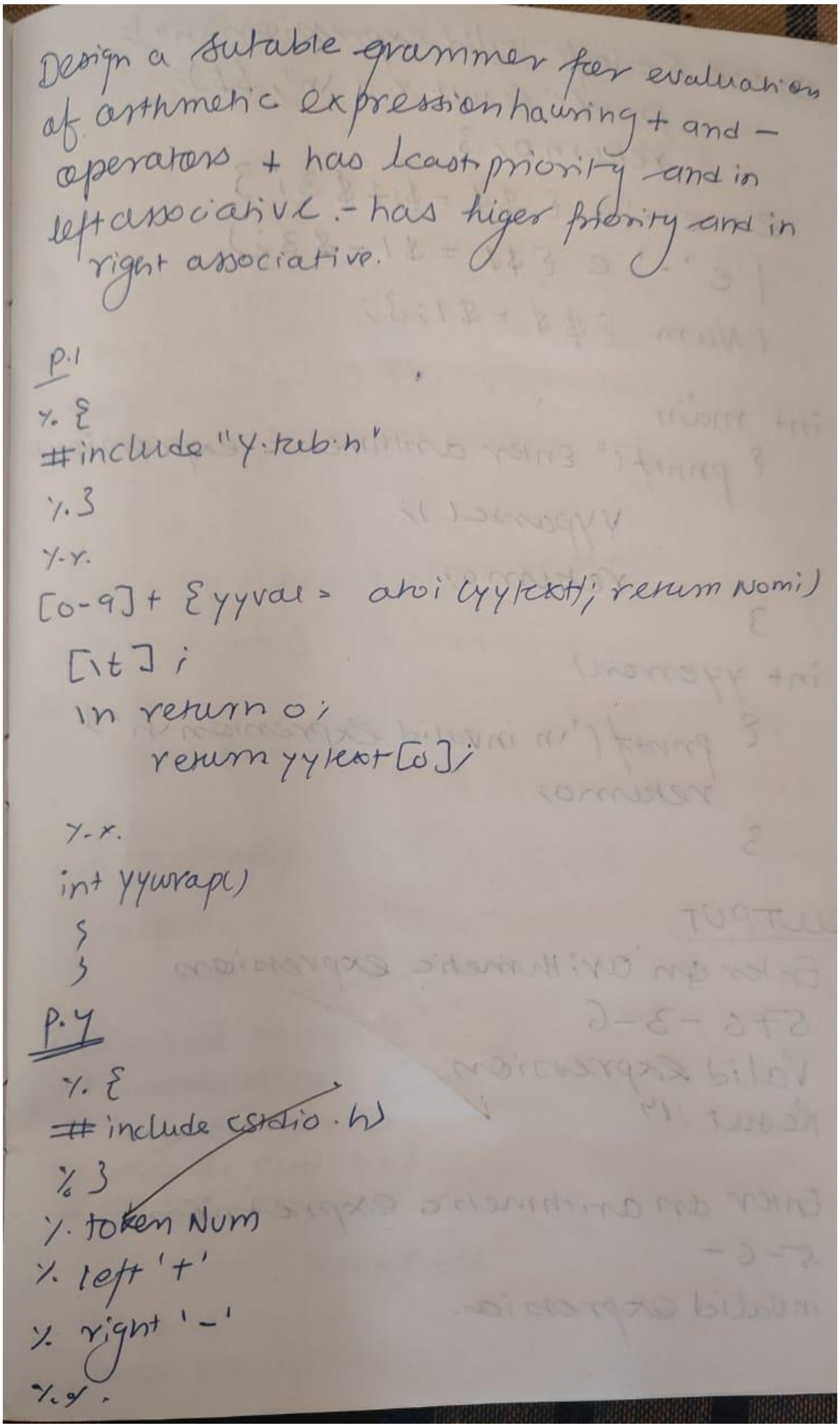


```
Enter a string:
cabd$
Valid string!
```

Lab 7

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

Code



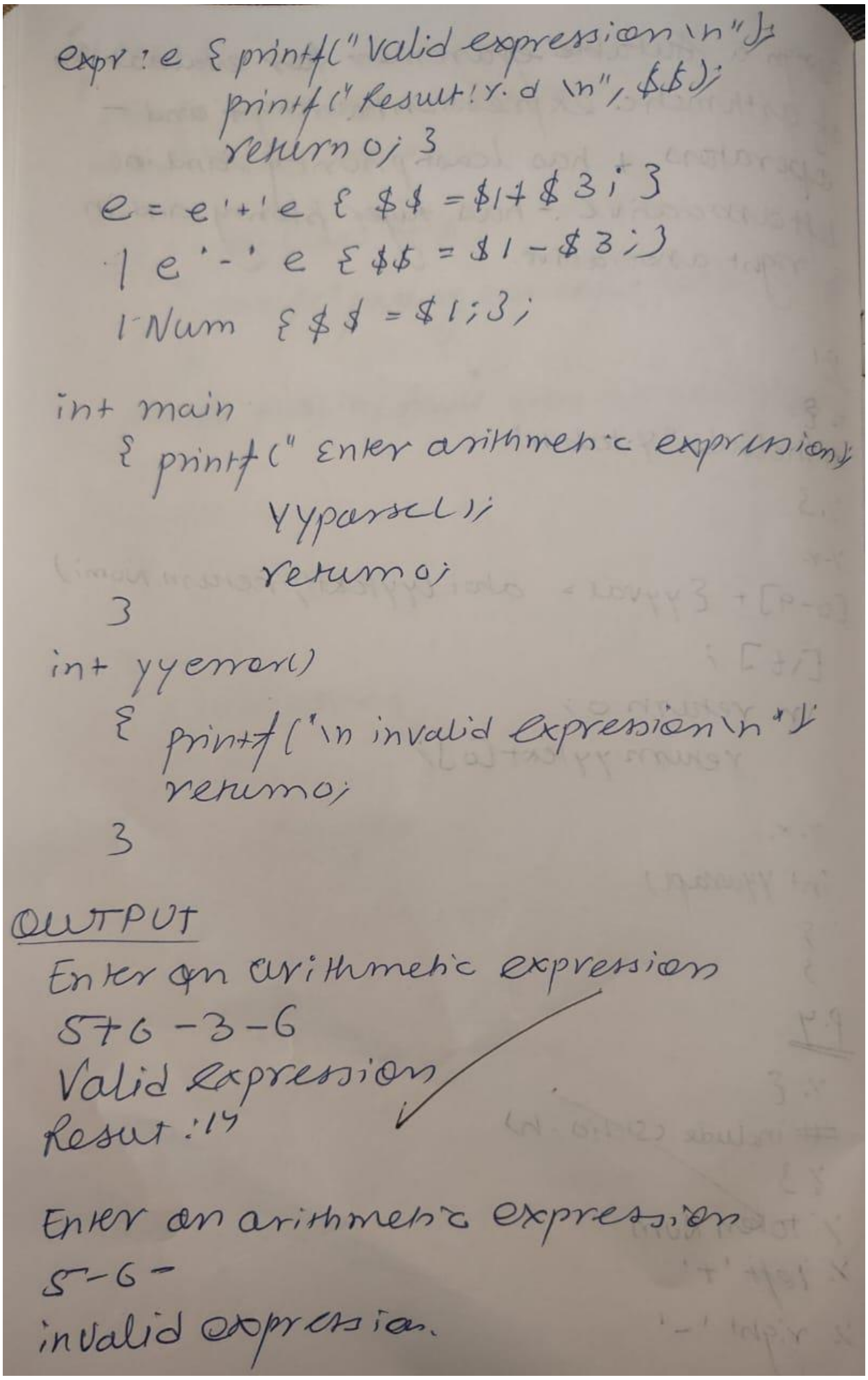
Design a Sutable grammer for evaluation of arthmetic expression having + and - operators + has least priority and in left associative. - has higer priority and in right associative.

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 '-'
%%.
```

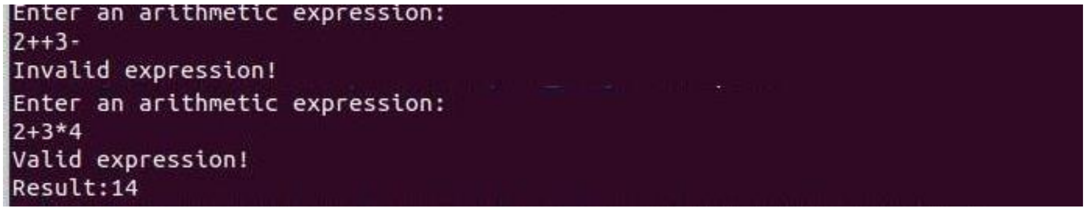


```
expr : e {printf("Valid expression\n");
          printf("Result:%d \n", $$);
          return 0; }
  e = e'+'e { $$ = $1+$3; }
  | e'-' e {$$ = $1 - $3;}
  | Num {$$ = $1;};

int main
  { printf(" Enter arithmetic expression");
          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.
```

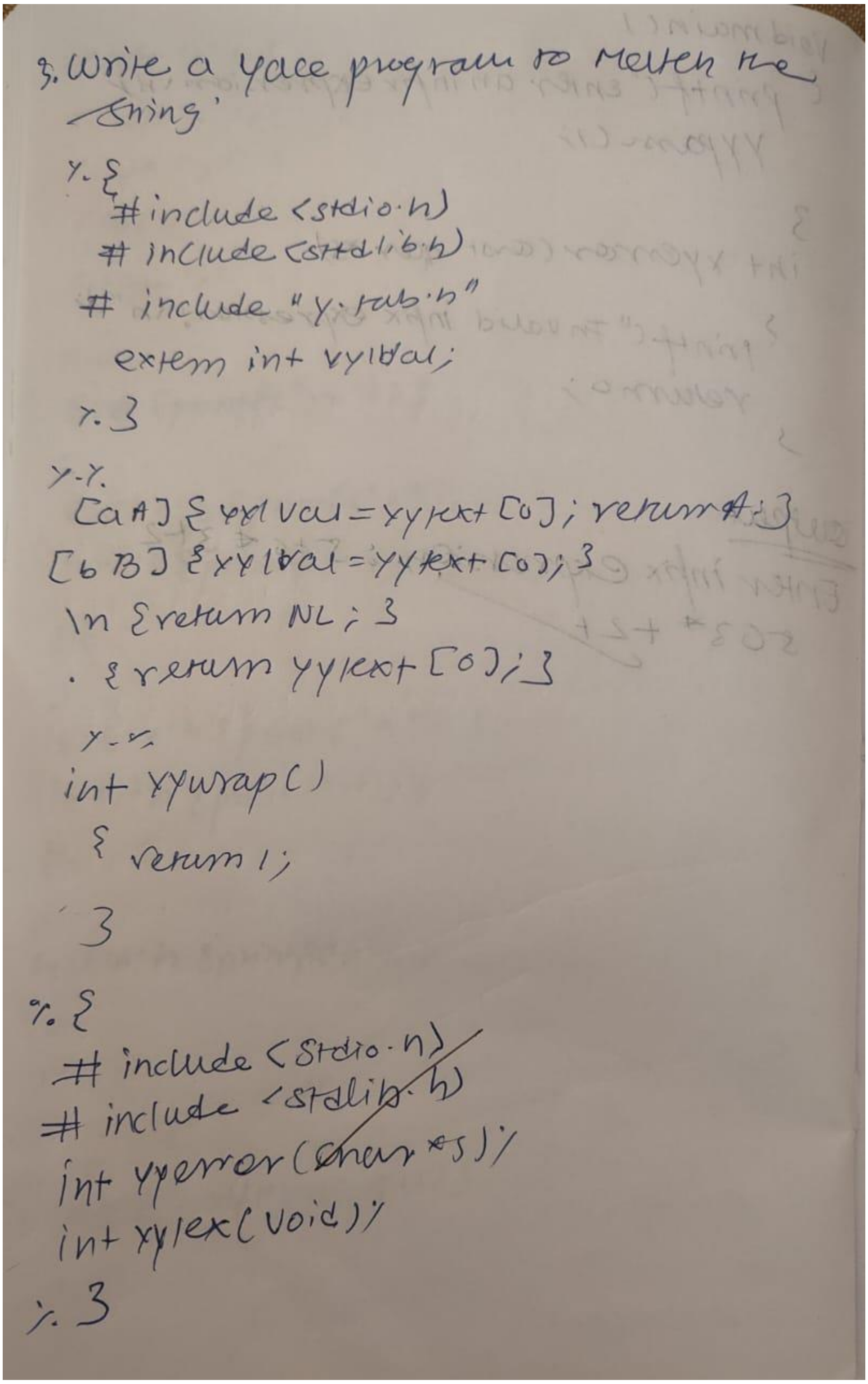
Output



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

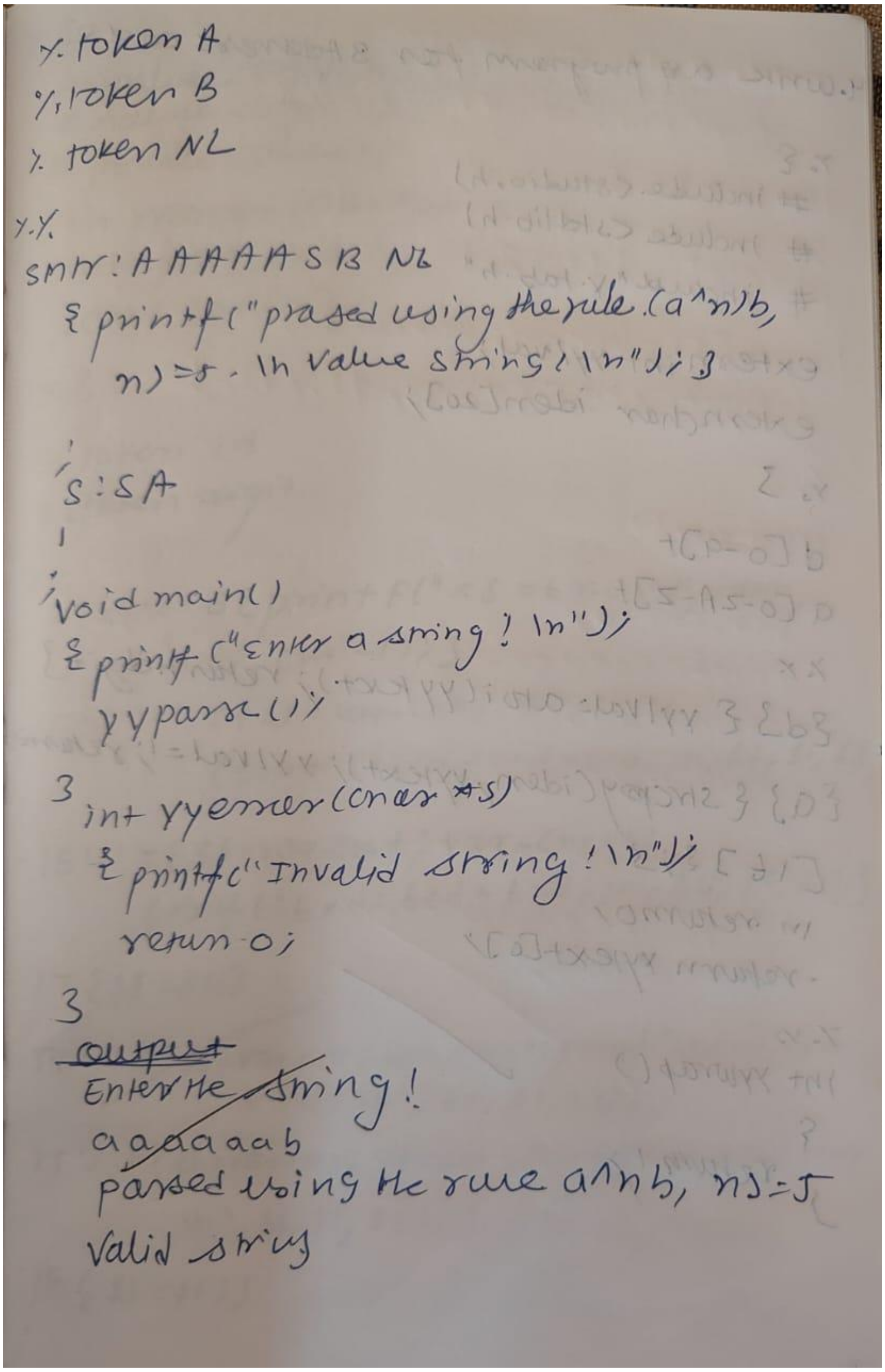
7.2 Write a program in YACC to recognize strings of the form {(a^n)b , n>=5}.

Code



3. Write a yacc program to match the string'

```
%{
  #include <stdio.h>
 #include <stdlib.h>
#include "y.tab.h"
  extern int yylval;
%}

%%
 [aA] {yylval=yytext[0]; return A;}
[bB] {yylval=yytext[0];}
 \n {return NL;}
 . {return yytext[0];}

 %%
int yywrap()
 { return 1;
 }

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



```
%.token A
%,token B
% token NL
%.%
stmt: A A A A A S B NL
 {printf("prased using the rule (a^n)b,
   n)=5. \n value string!\n");}
;
S:SA
|
;
void main()
{printf("Enter a string!\n");
 yyparse();
}
int yyerror(char *s)
{printf("Invalid string!\n");
 return 0;
}
```

~~output~~

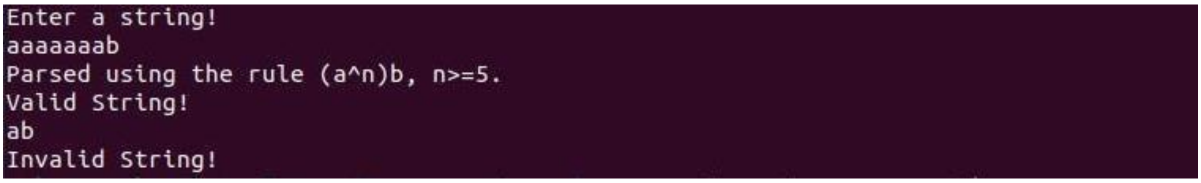
~~Enter the string!~~

~~aaaaaab~~

parsed using the rule a^nb, n>=5

Valid string

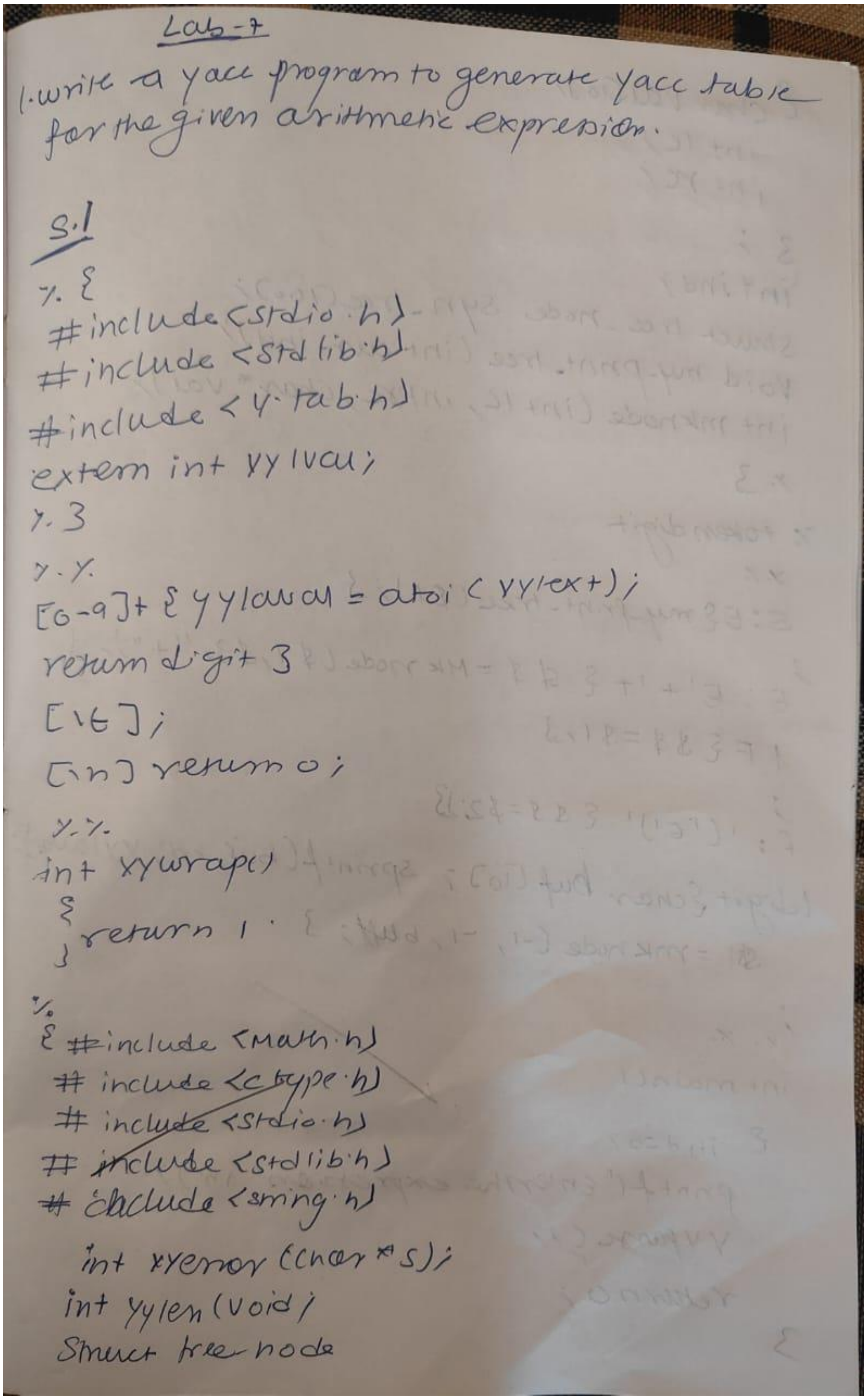
Output



```
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

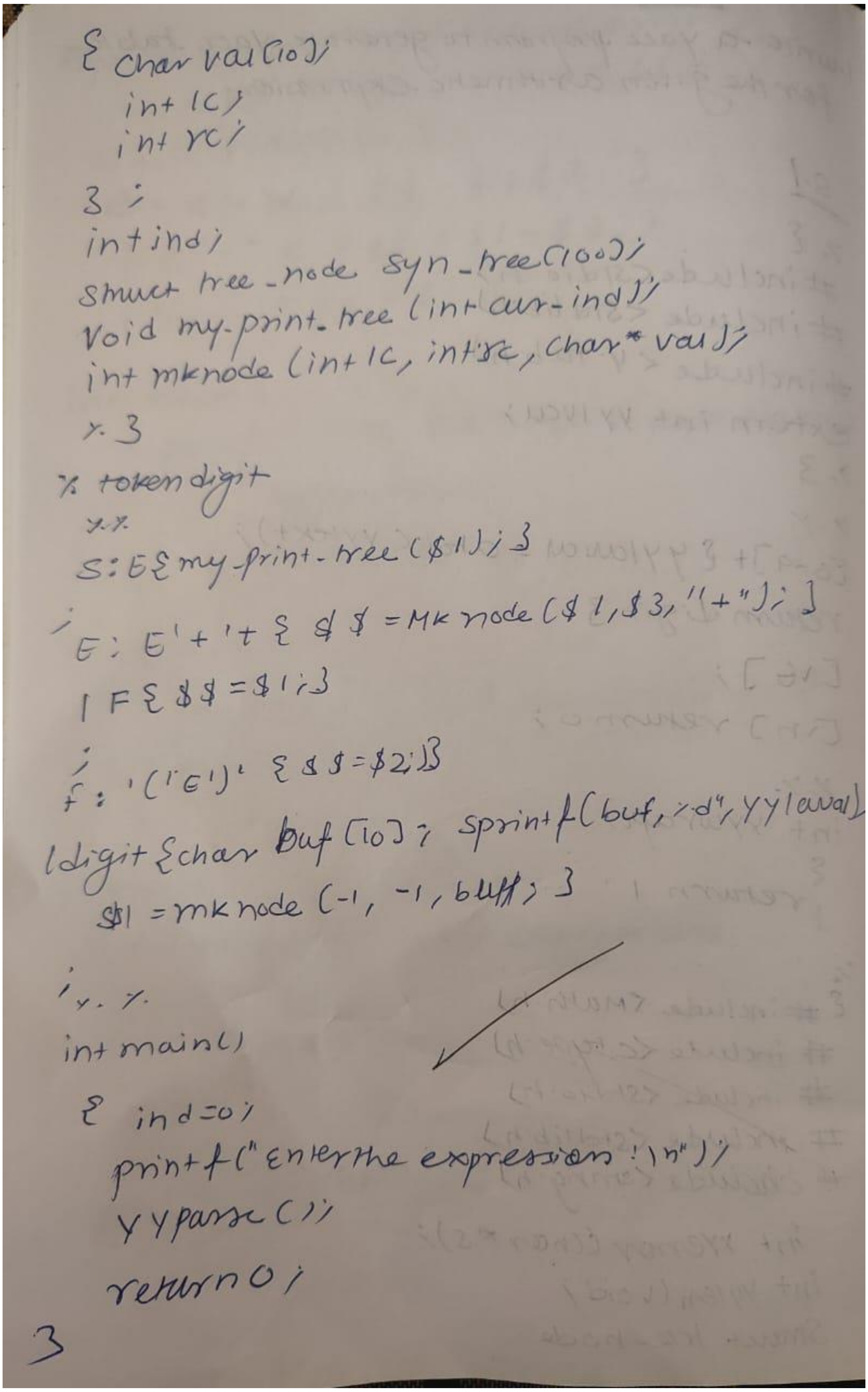


Lab-7

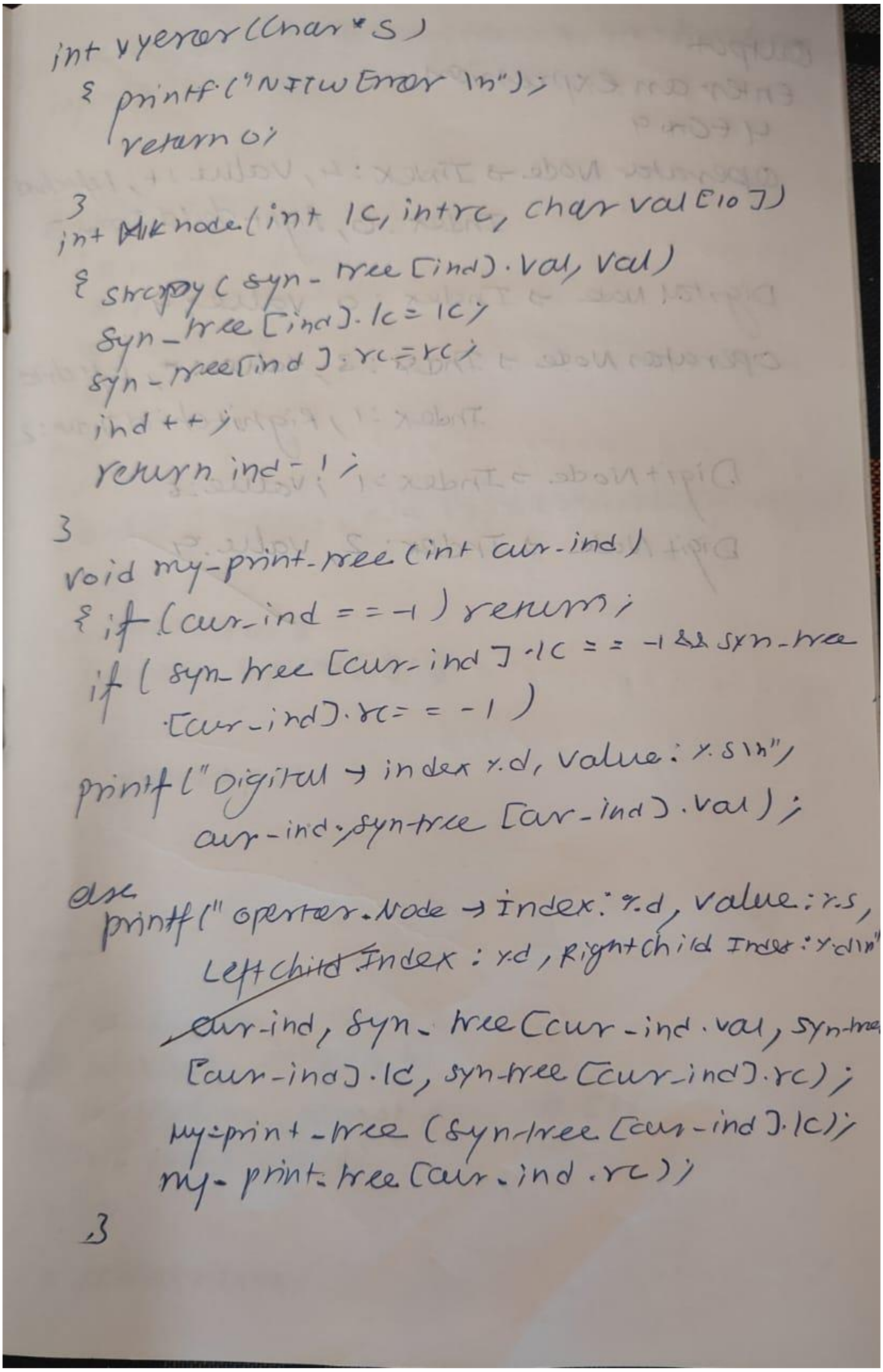
1. write a yacc program to generate yacc table for the given arithmetic expression.

S.1

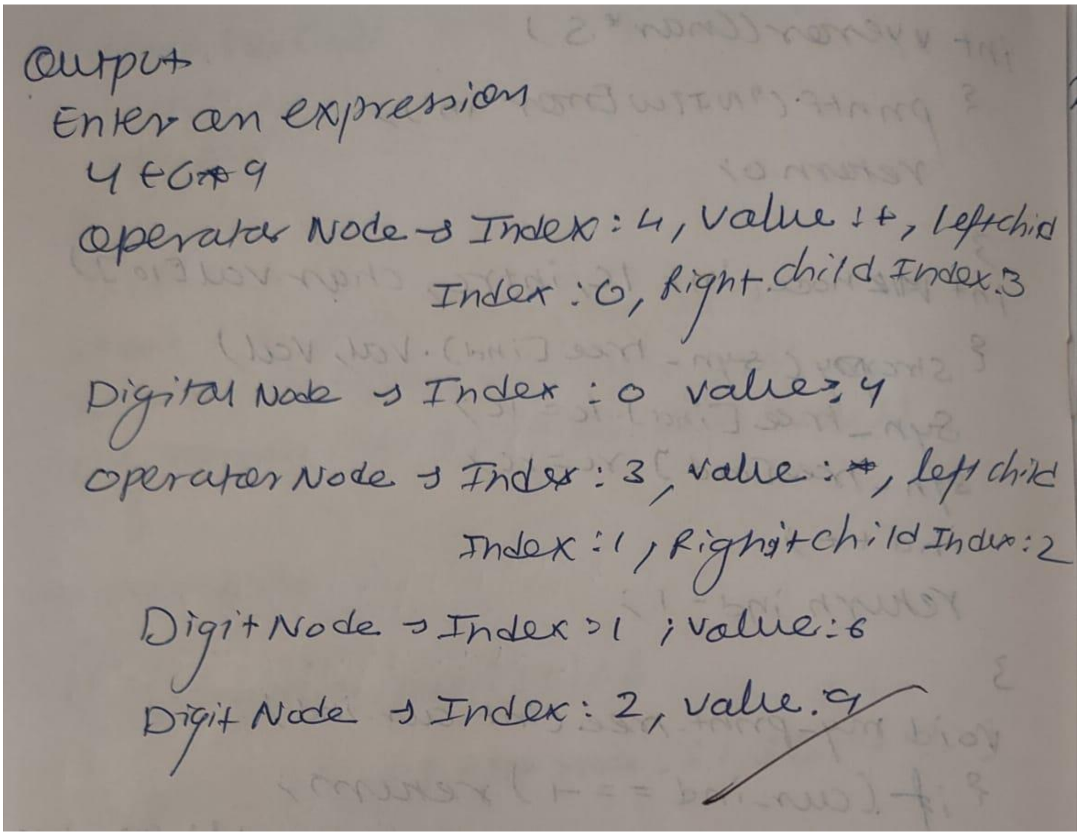
```
%{
 #include<stdio.h>
#include <stdlib.h>
#include < y.tab.h>
extern int yylval;
%}
%.%
[0-9]+ { yylaval = atoi ( yylext+);
return digit }
[\t];
[\n] return 0;
 %%
int xywrap()
 {
  return 1.
 }

%
{ #include <Math.h>
 # include <ctype.h>
 # include <stdio.h>
# include <stdlib.h>
# include <string.h>
  int xyerror (char *s);
 int yylen(void/
 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);
 %}
% token digit
 %%
 S: E{my_print_tree ($1);}
;
 E: E'+'t { $$ =Mk node ($1,$3,"+"); }
 | F{$$=$1;}
;
f: '('E')' {$$=$2;}}
|digit{char buf[10]; sprintf(buf,"%d",yylval)
  $1 =mknode (-1, -1, buff); }

;
%%
int main()
 { ind=0;
  printf("Enterthe expression :\n");
  yyparse();
  return0;
}
```

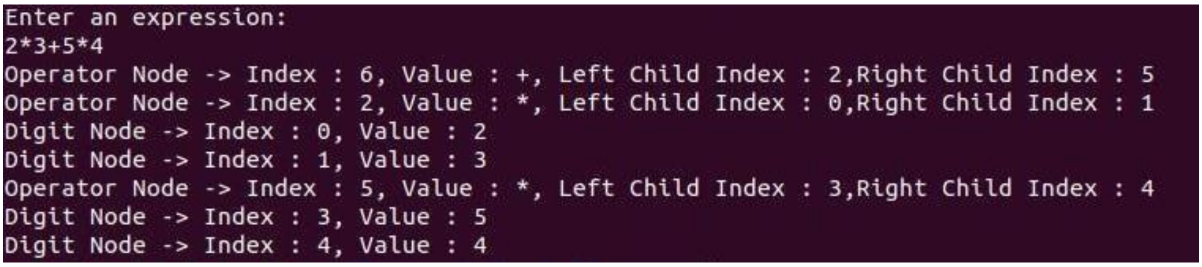


```
int yyerror(char *s)
  { printf("NITW Error \n");
    return 0;

}
int ~~Mk~~ node(int lc, int rc, char val[10])
 { strcpy(syn-tree[ind].val, val)
   syn_tree[ind].lc = lc;
   syn-tree[ind].rc = rc;
   ind++;
   return ind-1;

}
void my-print-tree(int cur-ind)
 { if (cur_ind == -1) return;
   if ( syn-tree [cur-ind].lc == -1 && syn-tree
         [cur-ind].rc == -1)
   printf("Digital → index %d, value : %s\n",
          cur-ind, syn-tree [cur-ind].val);

  else
   printf("operter.Node → index: %d, value: %s,
          LeftChild Index : %d, RightChild Index: %d\n"
          cur-ind, syn-tree[cur-ind.val, syn-tree
          [cur-ind].lc, syn-tree[cur_ind].rc);
          my-print-tree (syn-tree [cur-ind].lc);
          my-print-tree(cur.ind.rc));
  }
```



```
Output
 Enter an expression
  4+6*9
  Operator Node → Index: 4, value :+, leftchild
                    Index : 0, Right child Index:3

  Digital Node → Index : 0  value: 4
  Operator Node → Index : 3, value : *, left child
                     Index : 1, Right child Index: 2
   Digit Node → Index : 1 ; value: 6
   Digit Node → Index: 2, value: 9
```

Output

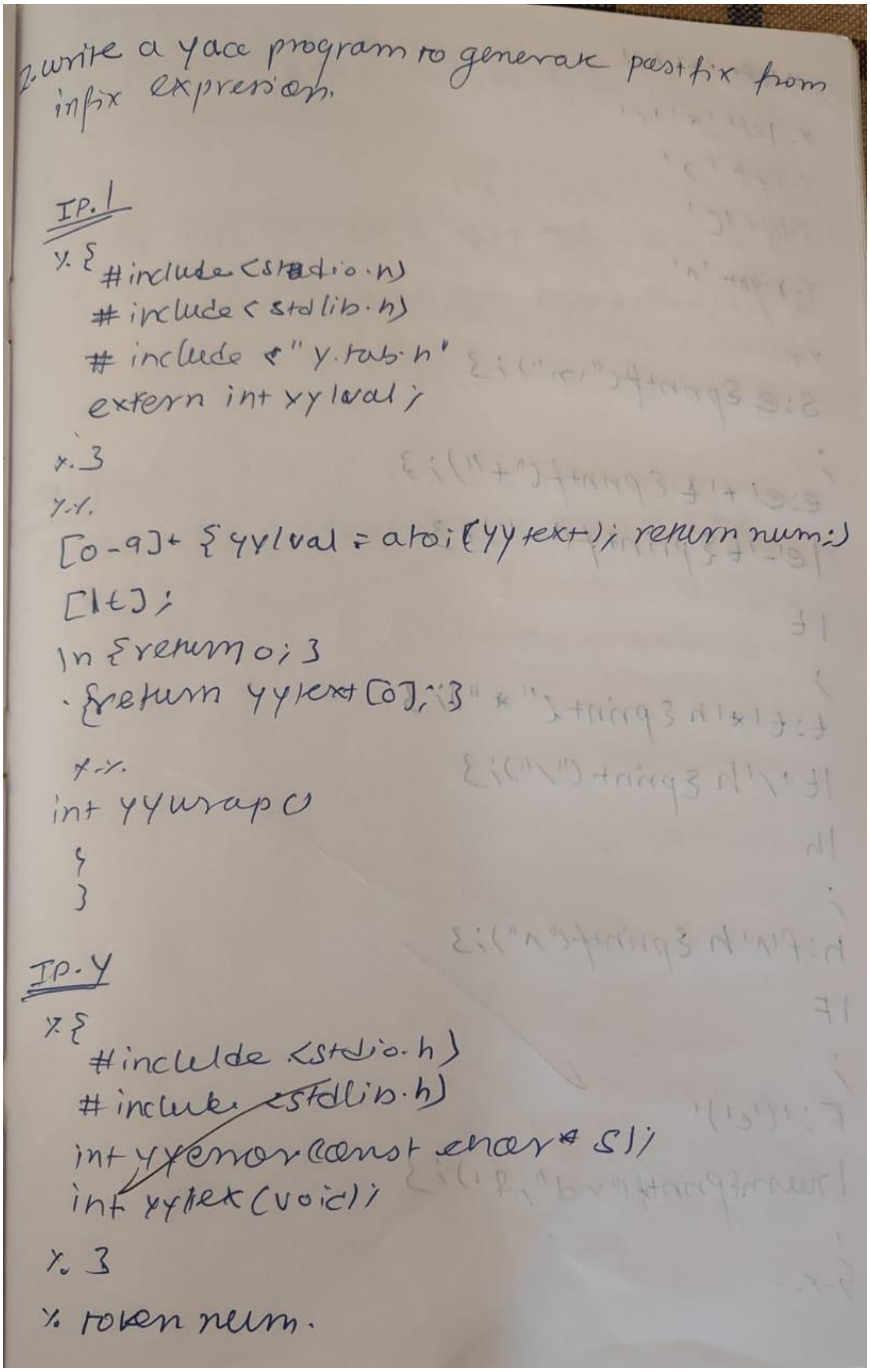


```
Enter an expression:
2*3+5*4
Operator Node -> Index : 6, Value : +, Left Child Index : 2,Right Child Index : 5
Operator Node -> Index : 2, Value : *, Left Child Index : 0,Right Child Index : 1
Digit Node -> Index : 0, Value : 2
Digit Node -> Index : 1, Value : 3
Operator Node -> Index : 5, Value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



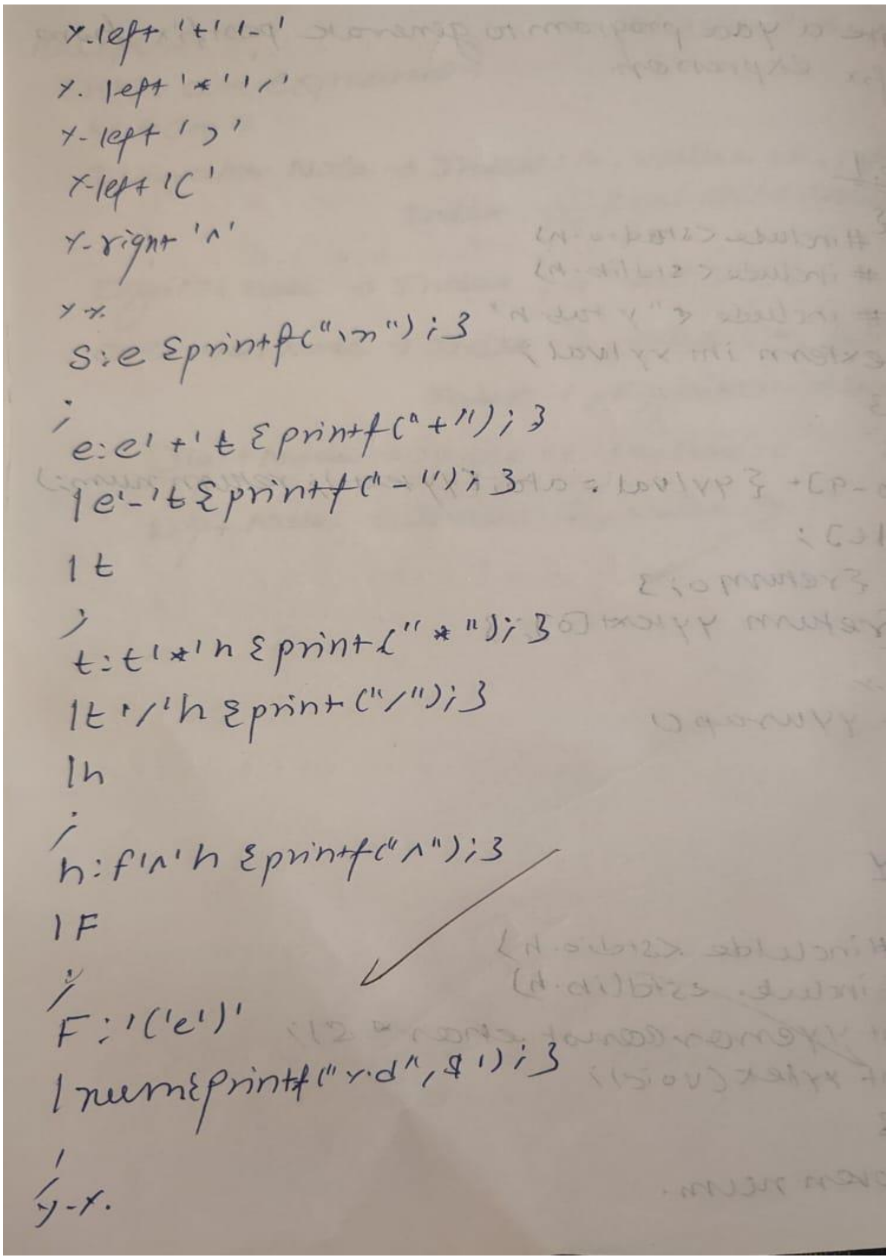
2. write a yacc program to generate postfix from infix expression.

IP.l

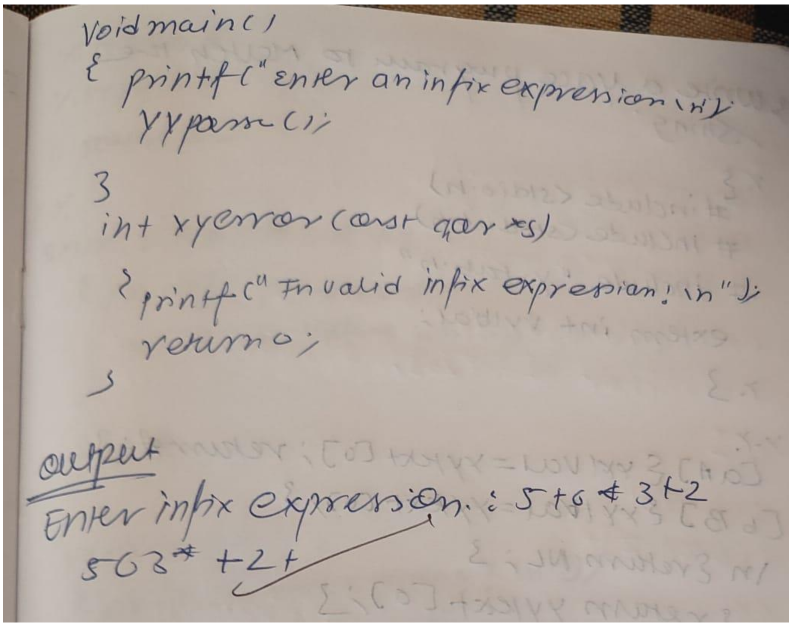
```
%{ #include <stdio.h>
   #include <stdlib.h>
   # include "y.tab.h"
   extern int yylval;

%}
%%
[0-9]+ {yylval = atoi(yytext); return num;}
[\t];
\n {return 0;}
. {return yytext[0];}

%%
int yywrap()
 {
 }
```

IP.y

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



```
%left '+' '-'
%. left '*' '/'
%- left ')'
%-left '('
%-right '^'

%%
S:e {printf("\n");}

;
e:e'+'t {printf("+");}
|e'-'t{printf("-");}
|t
;
t:t'*'h {printf("*");}
|t'/'h {print("/");}
|h
;
h:f'^'h {printf("^");}
|F
;
F:'('e')'
|num{printf("%d",$1);}
;
%-%.
```

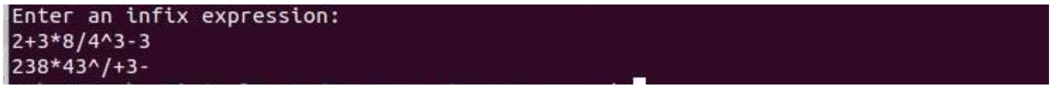


```
void main()
{ printf("enter an infix expression\n");
  yyparse();

}
int yyerror(const char *s)
 {printf("Invalid infix expression!\n");
  return0;
 }
```

<u>output</u>

```
Enter infix expression : 5+6*3+2
 563*+2+
```

Output

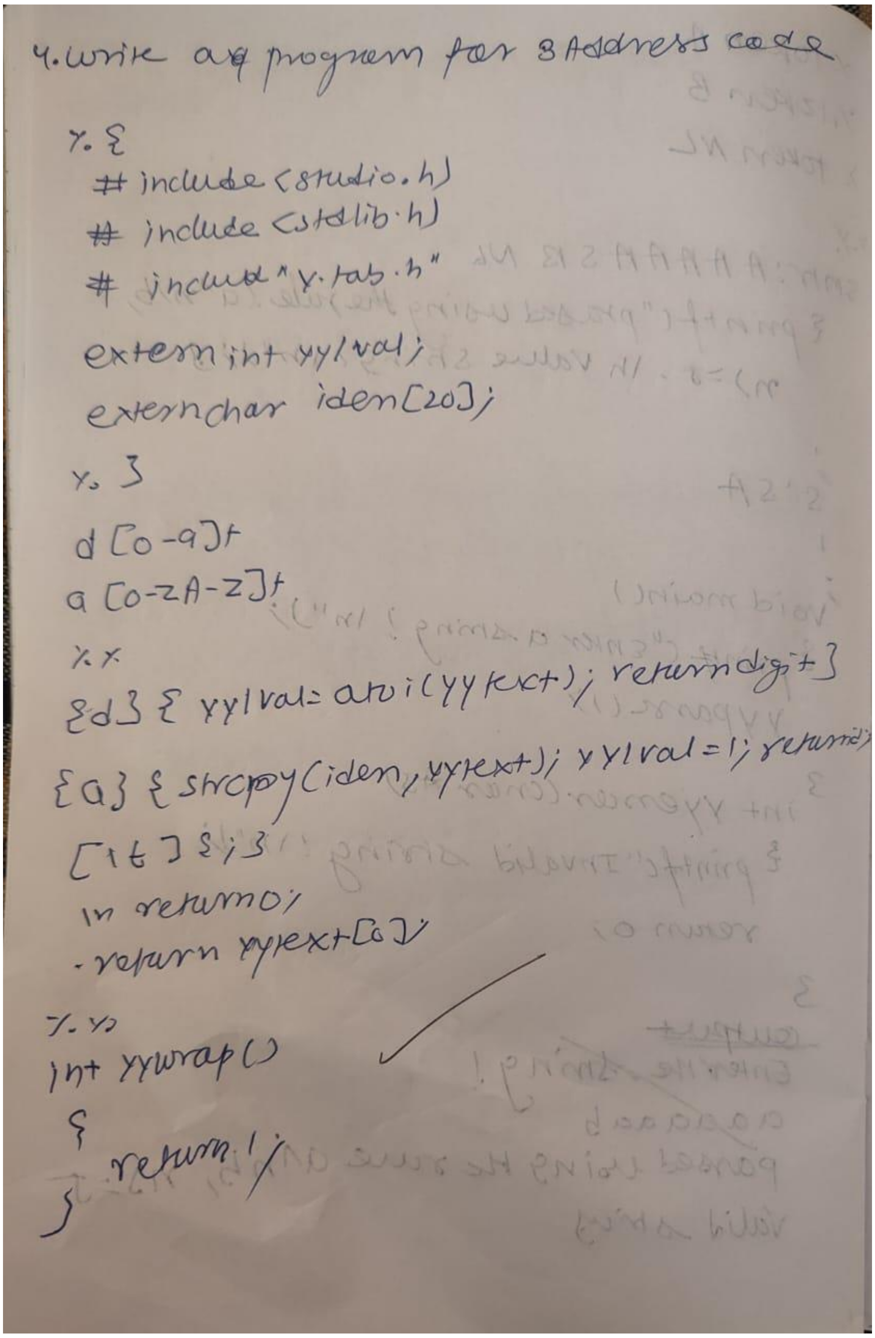


```
Enter an infix expression:
2+3*8/4^3-3
238*43^/+3-
```

Lab 9

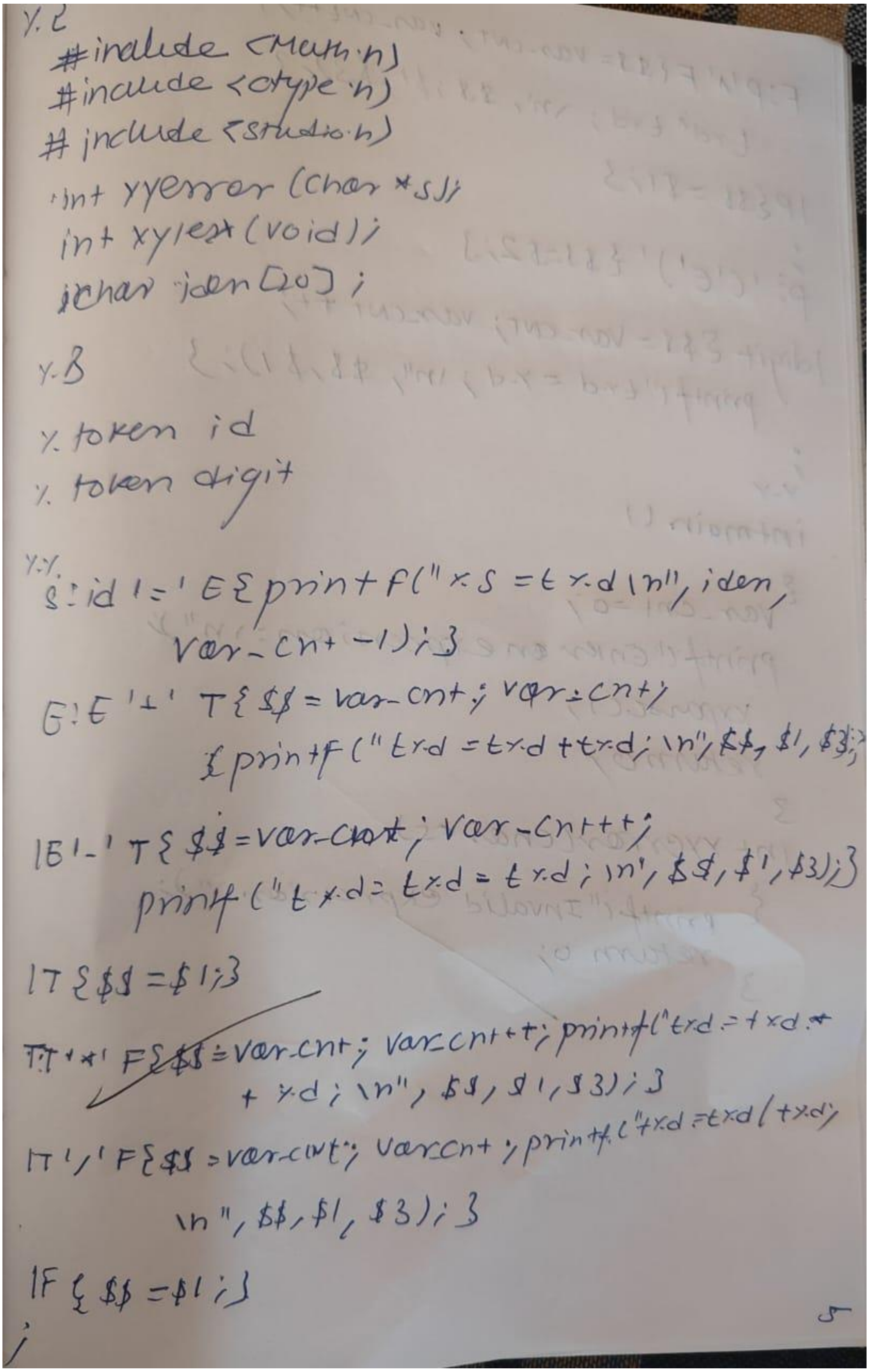
9.1 Write a program in YACC to generate three address code for a given expression.

Code



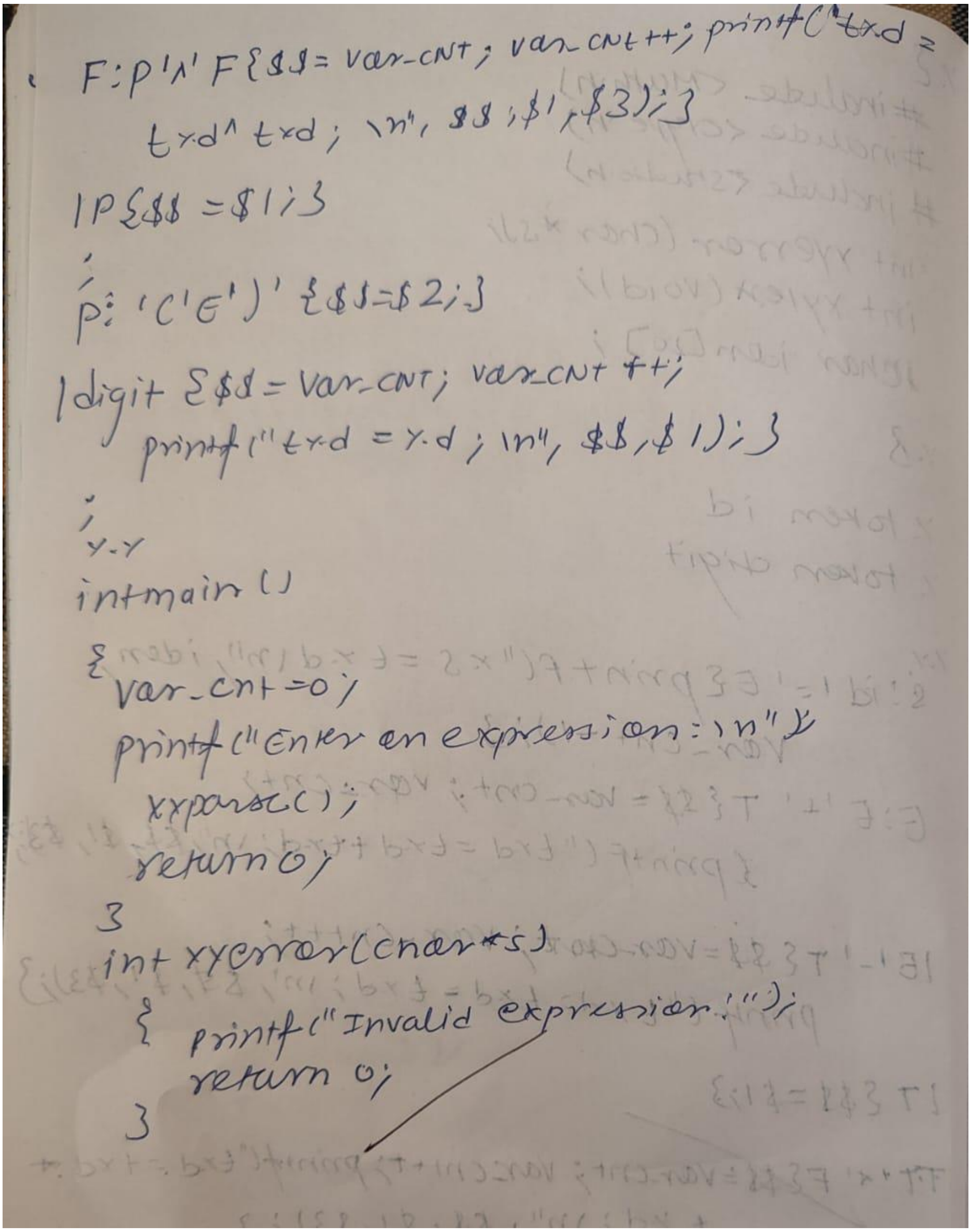
4. write a~~q~~ program for 3 Address code

```
%{
 # include <studio.h>
 # include <stdlib.h>
 # include "y.tab.h"
 extern int yylval;
 externchar iden[20];
%}
 d [0-9]+
 a [0-zA-z]+
%%
{d} { yylval= atoi(yytext); returndigit }
{a} { strcpy(iden, yytext); yylval=1; returnid;
 [\t ] {;}
 \n return0;
 . return yytext[0];
%%
int yywrap()
 {
  return 1;
}
```

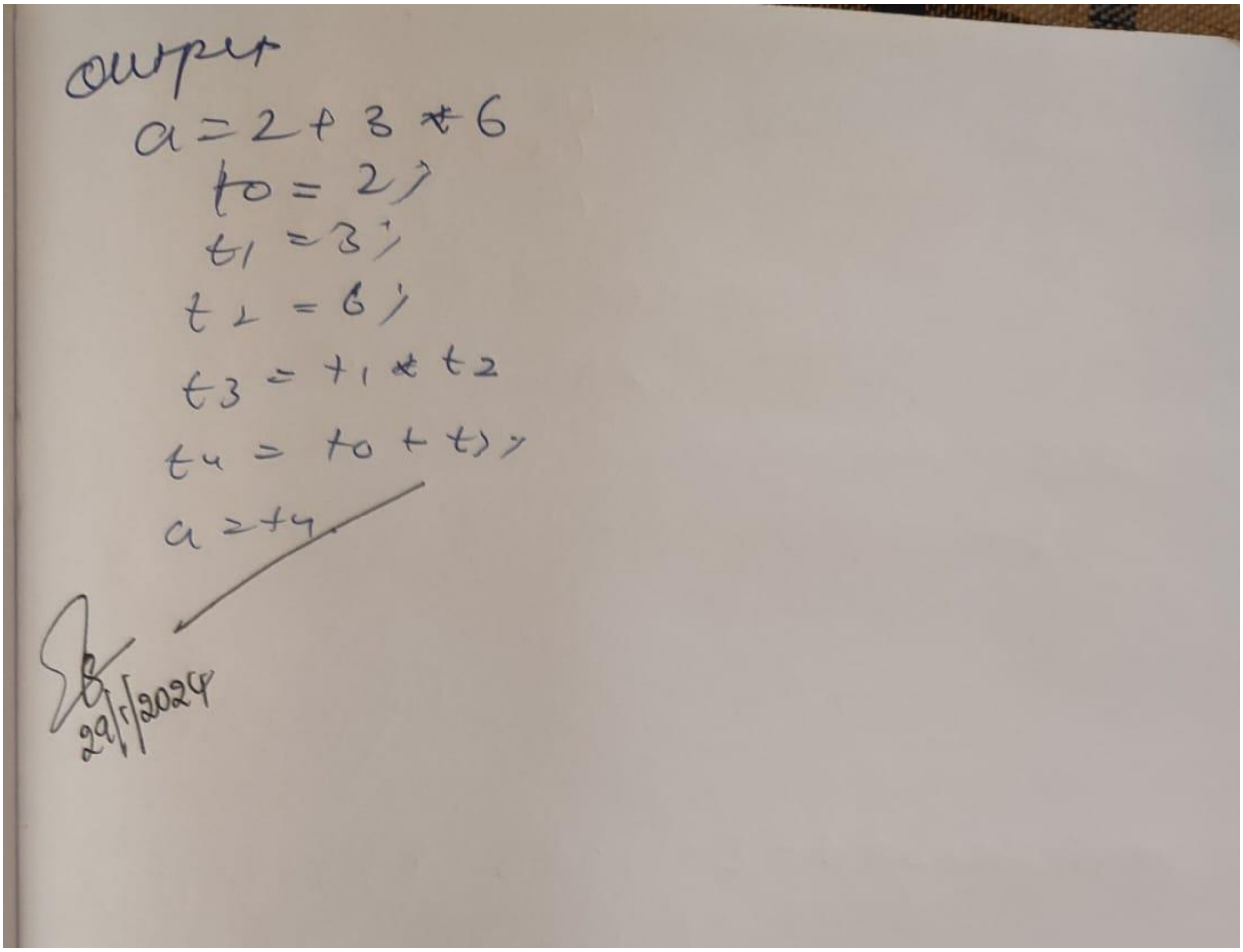


```
%{
 #include <Math.h>
 #include <ctype.h>
 #include <studio.h>
 int yyerror (char *s);
 int xylex (void);
 char iden[20];

%}

%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", $$, $1, $3);}
|T '/' F{ $$ = var_cnt; var_cnt; printf("t%d = t%d / t%d;
        \n", $$, $1, $3);}
|F{ $$ = $1;}
;
```

5



```
F:P'^'F{$$= var_cnt; var_cnt++; printf("t%d =
    t%d^t%d; \n", $$,$1,$3);}
|P{$$ =$1;}
;
P: '('E')' {$$=$2;}
|digit {$$= var_cnt; var_cnt++;
    printf("t%d = %d; \n", $$,$1);}
;
%%
intmain()
{ var_cnt=0;
  printf("Enter en expression:\n");
   yyparse();
  return 0;
 }
 int yyerror(char*s)
  { printf("Invalid expression!");
    return 0;
  }
```

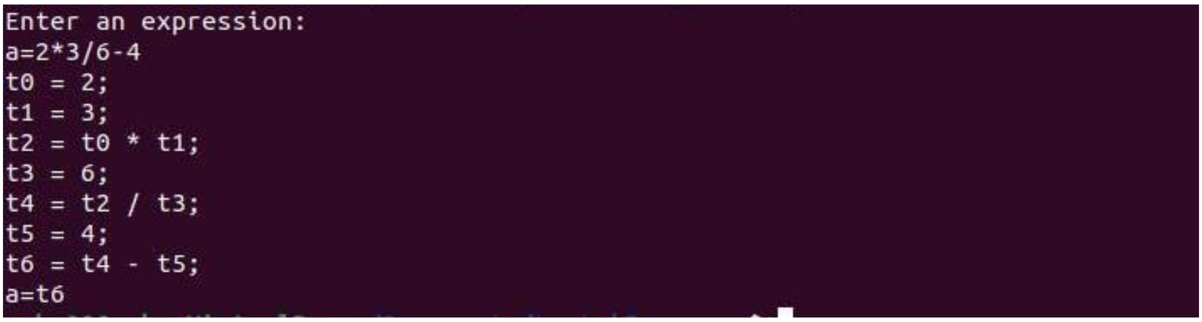


output

```
a = 2 + 3 * 6
t0 = 2;
t1 = 3;
t2 = 6;
t3 = t1 * t2
t4 = t0 + t3;
a = t4.
```

29/1/2024

Output



```
Enter an expression:
a=2*3/6-4
t0 = 2;
t1 = 3;
t2 = t0 * t1;
t3 = 6;
t4 = t2 / t3;
t5 = 4;
t6 = t4 - t5;
a=t6
```