VISVESVARAYA TECHNOLOGICAL UNIVERSITY

"JnanaSangama", Belgaum -590014, Karnataka.

LAB REPORT

on

COMPILER DESIGN

Submitted by

IBRAHIM IFTEKHAR KHAN (1BM21CS076)

Under the Guidance of

Prof. Prameetha Pai
Assistant Professor,
BMSCE

in partial fulfillment 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**" was carried out by **Ibrahim Iftekhar Khan (1BM21CS076)**, who is a bonafide student of **B. M. S. College of Engineering.** It is in partial fulfillment 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 BMSCE, Bengaluru BMSCE, Bengaluru

B. M. S. COLLEGE OF ENGINEERING

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

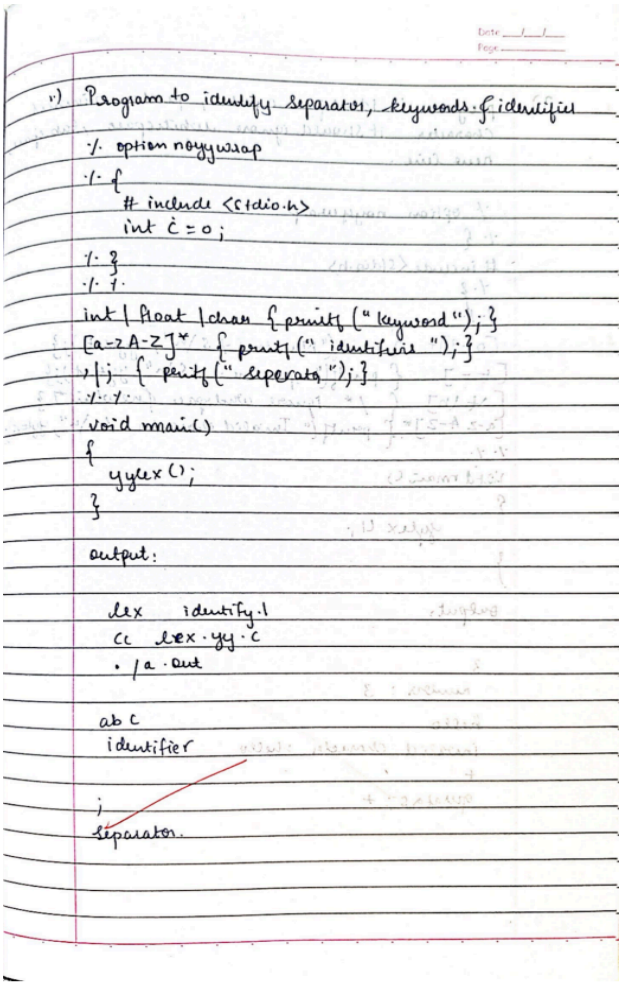
DECLARATION

I, Ibrahim Iftekhar Khan (1BM21CS076), student of 5th Semester, B.E, Department of Computer Science and Engineering, B. M. S. College of Engineering, Bangalore, hereby declare that this lab report entitled " **Compiler Design**" has been carried out by me under the guidance of Prof. Latha NR, Assistant Professor, Department of CSE, B. M. S. College 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 part of any other report by any other students.

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

Code:



1) Program to identify separator, keywords & identifier

```
%{
    #include <stdio.h>
    int c = 0;
%}
%%
int | float | char {printf("keyword");}
[a-zA-Z]* {printf(" identifier ");}
;|, {printf(" seperator ");}
%%
void main()
{
  yylex();
}
```

output:

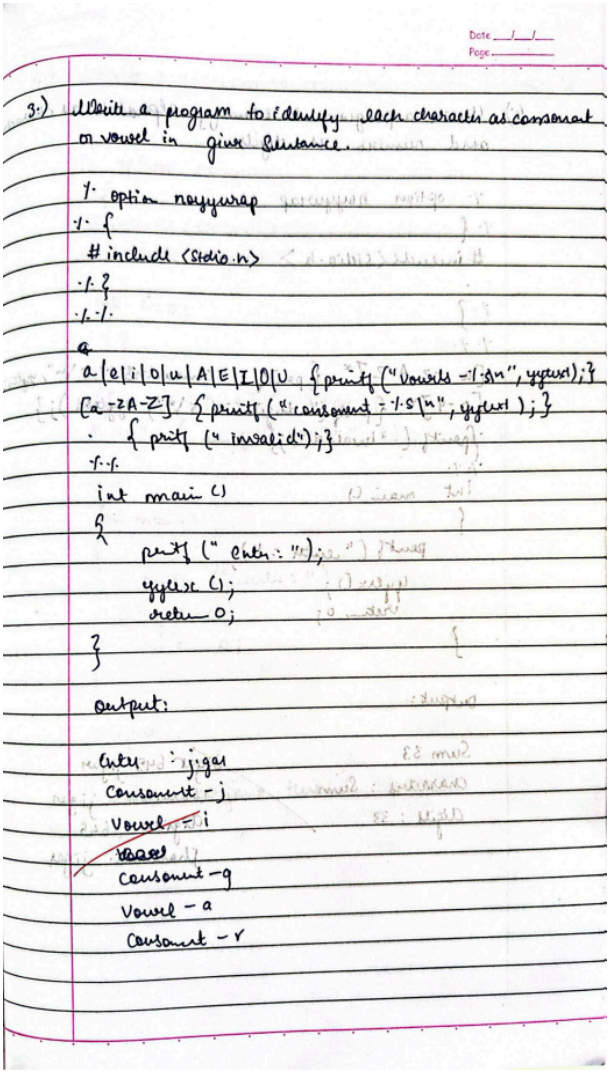
```
lex identify.l
cc lex.yy.c
./a.out

abc
identifier

;
seperator.
```

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
```

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

Code



3.) Write a program to identify each character as consonant or vowel in given sentence.

```
%option noyywrap
%{
#include <stdio.h>
%}
%%
a
a|e|i|o|u|A|E|I|O|U {printf("Vowels - %s\n",yytext);}
[a-zA-Z] {printf("consonant - %s\n",yytext);}
. {printf("invalid");}
%%
int main()
{
    printf("enter:");
    yylex();
    return 0;
}
```

Output:

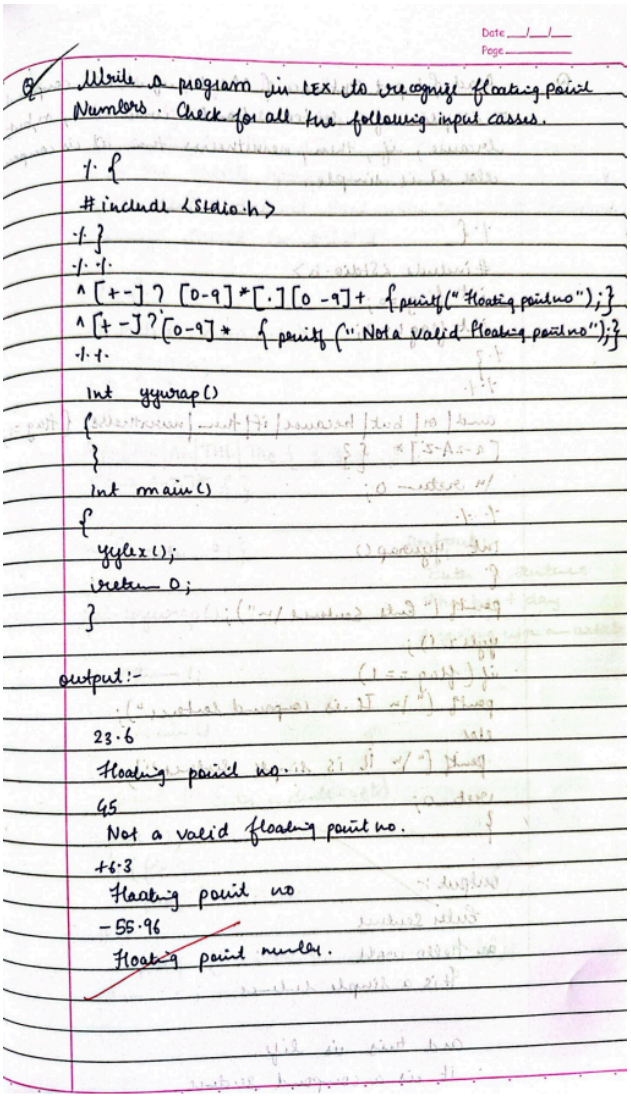
```
Enter : jigar
Consonant - j
Vowel - i
Consonant - g
Vowel - a
Consonant - r
```

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
```

Write a program in LEX to recognize Floating Point Numbers.

Code



Write a program in LEX to recognize floating point Numbers. Check for all the following input cases.

```
%{
#include<stdio.h>
%}
%%
^[+-]?[0-9]*[.][0-9]+ {printf("Floating point no");}
^[+-]?[0-9]* {printf("Not a valid floating point no");}
%%
int yywrap()
{
}
int main()
{
yylex();
return 0;
}
```

output:-

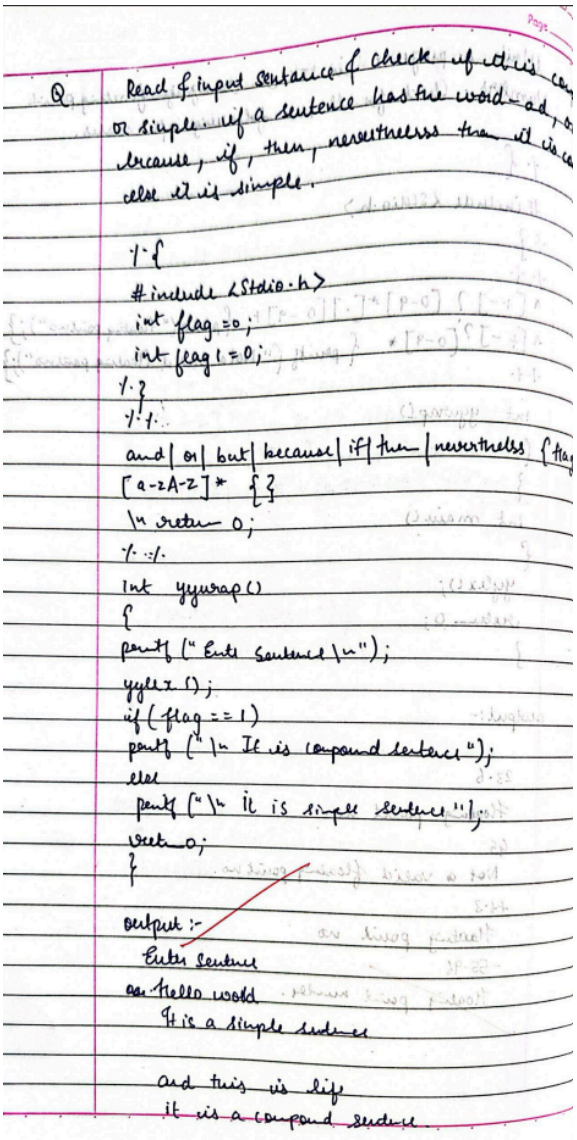
23.6
Floating point no.
45
Not a valid floating point no.
+6.3
Floating point no
-55.96
Floating point number.

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!
```

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 & input sentence & check if it is co
or simple. if a sentence has the word and, o
because, if, then, nevertheless then it is c
else it is simple.

%{
#include <stdio.h>
int flag=0;
int flag1=0;
%}
%%
and | or | but | because | if | then | nevertheless {fla
[a-zA-Z]* {}
\n return 0;
%%
int yywrap()
{
printf("Enter sentence\n");
yylex();
if(flag==1)
printf("\n It is compound sentence");
else
printf("\n it is simple sentence");
return 0;
}

output :-
Enter sentence
oh hello world
   It is a simple sentence

   and this is life
   it is a compound sentence
```

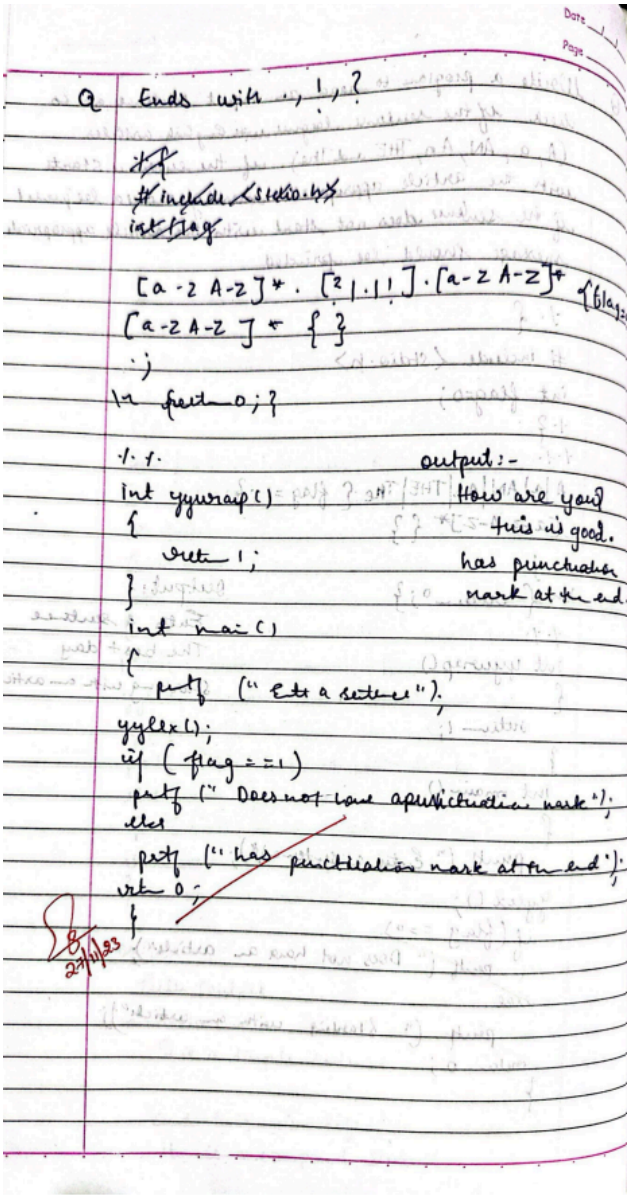
Output

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

```
Enter a sentence:
She is good at singing and dancing.
Compound sentence!
```

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

Code



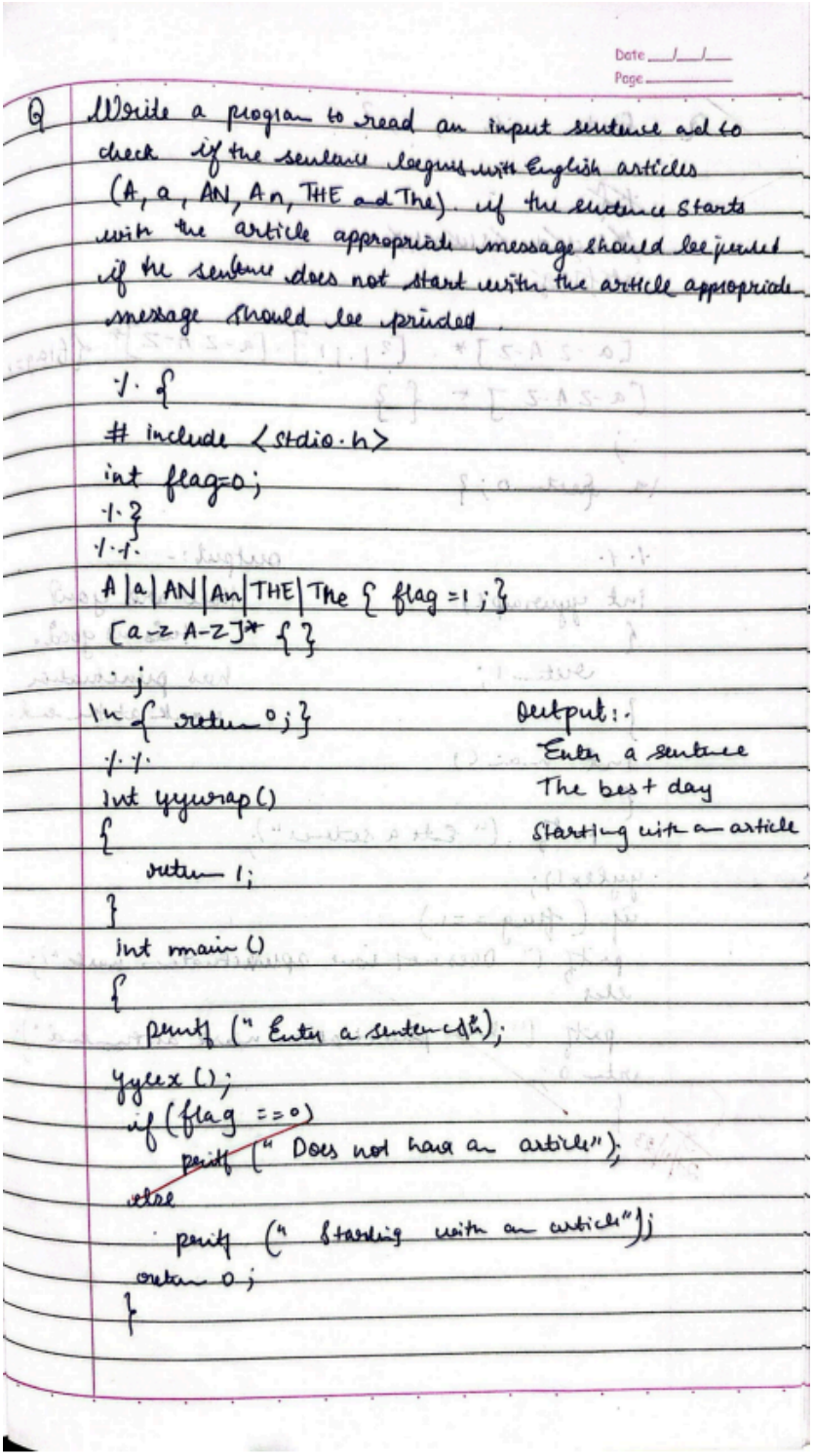
```
Q  Ends with . ! ?

#include<stdio.h>
int flag;

[a-z A-Z]*.[?|.|!].[a-z A-Z]* {flag=1;}
[a-z A-Z]* {}
.;
int main=0;?

/*.*/                                  output:-
int yywrap()                           How are you?
{                                      this is good.
  ret 1;                               has punctuation
}                                      mark at the end
int main()
{
  puts ("Enter a sentence");
yylex();
if (flag==1)
puts ("Does not have punctuation mark");
else
puts ("has punctuation mark at the end");
ret 0;
}
```

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!
```

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

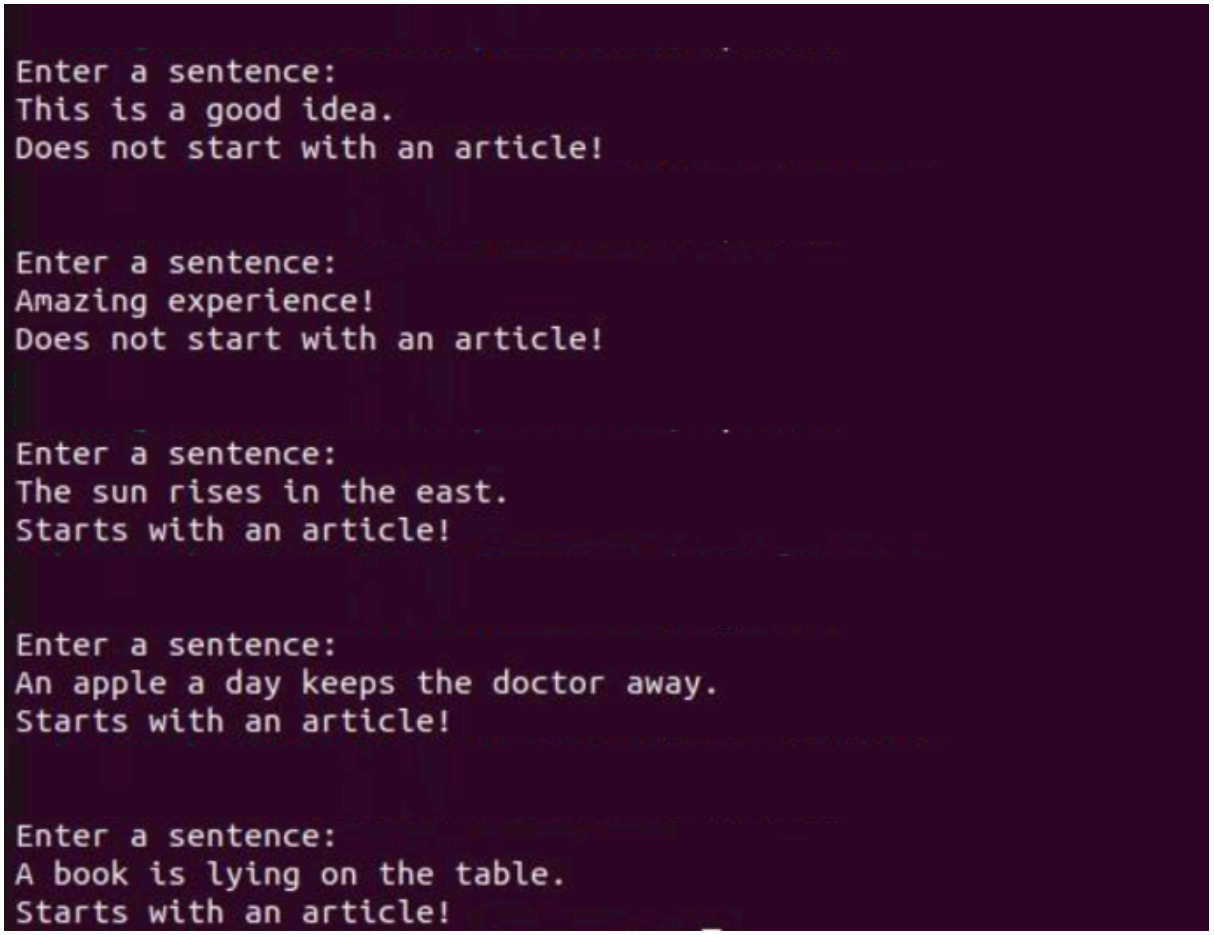


Q Write a program to read an input sentence and to check if the sentence begins with English articles (A, a, AN, An, THE and The). if the sentence starts with the article appropriate message should be printed if the sentence does not start with the article appropriate message should be printed.

```
%{
#include <stdio.h>
int flag=0;
%}
%%
A|a|AN|An|THE|The { flag=1;}
[a-z A-Z]* {}
;
\n{ return 0;}
%%
int yywrap()
{
  return 1;
}
int main()
{
  printf("Enter a sentence\n");
yylex();
if(flag==0)
  printf("Does not have an article");
else
  printf("Starting with an article");
return 0;
}
```

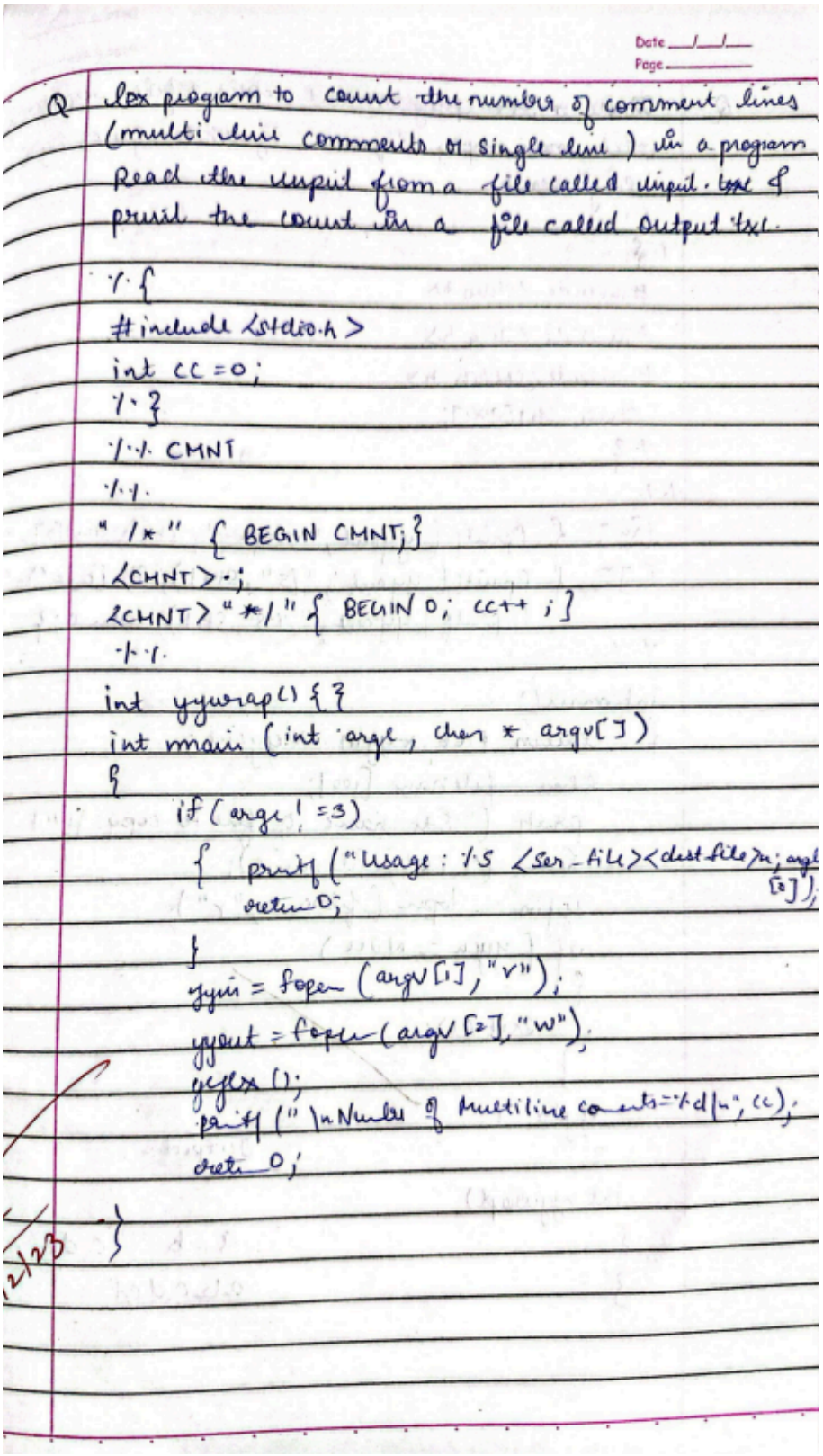
Output:-
Enter a sentence
The best day
Starting with an article

Output


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

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.

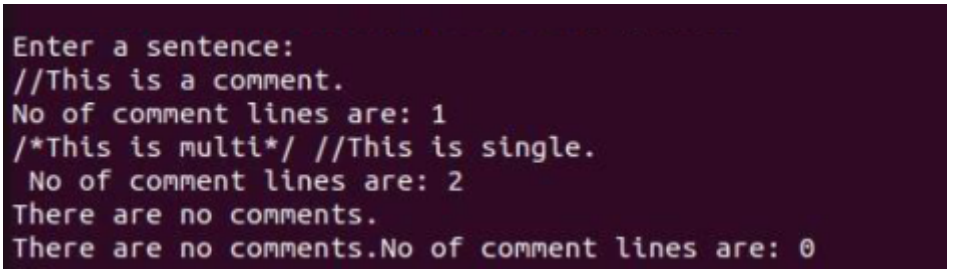
Code



Q 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 & print the count in a file called output.txt.

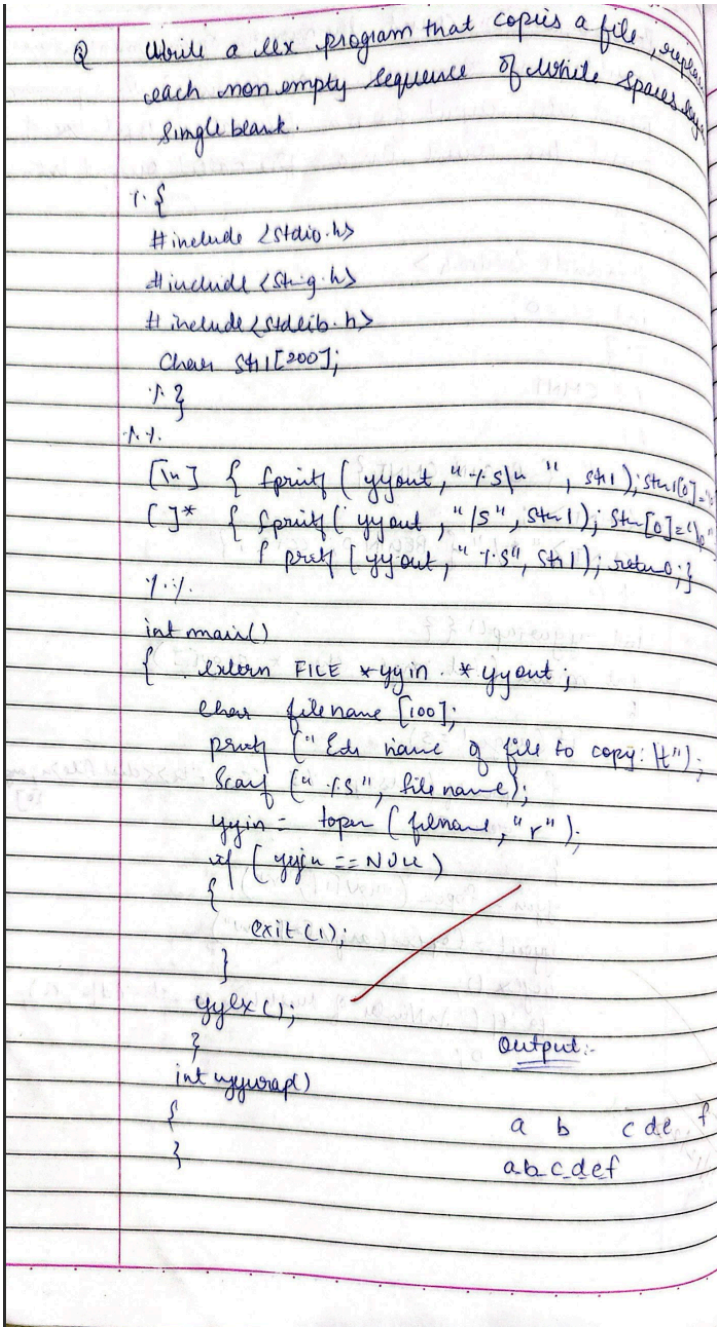
```
%{
#include<stdio.h>
int cc=0;
%}
%x CMNT
%%
"/*"  { BEGIN CMNT;}
<CMNT>.;
<CMNT>"*/" { BEGIN 0; cc++ ;}
%%
int yywrap() {}
int main (int argc, char * argv[])
{
    if(argc!=3)
    { printf("Usage: %s <src_file><dest file>\n",argv[0]);
      return 0;
    }
yyin = fopen (argv[1],"r");
yyout = fopen(argv[2],"w");
yylex();
printf("\n Number of multiline comments=%d\n",cc);
return 0;
}
```

Output

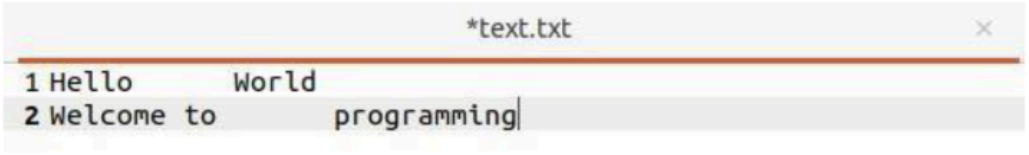

Enter a sentence:
//This is a comment.
No of comment lines are: 1
/*This is multi*/ //This is single.
No of comment lines are: 2
There are no comments.
There are no comments.No of comment lines are: 0

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

Code



Output

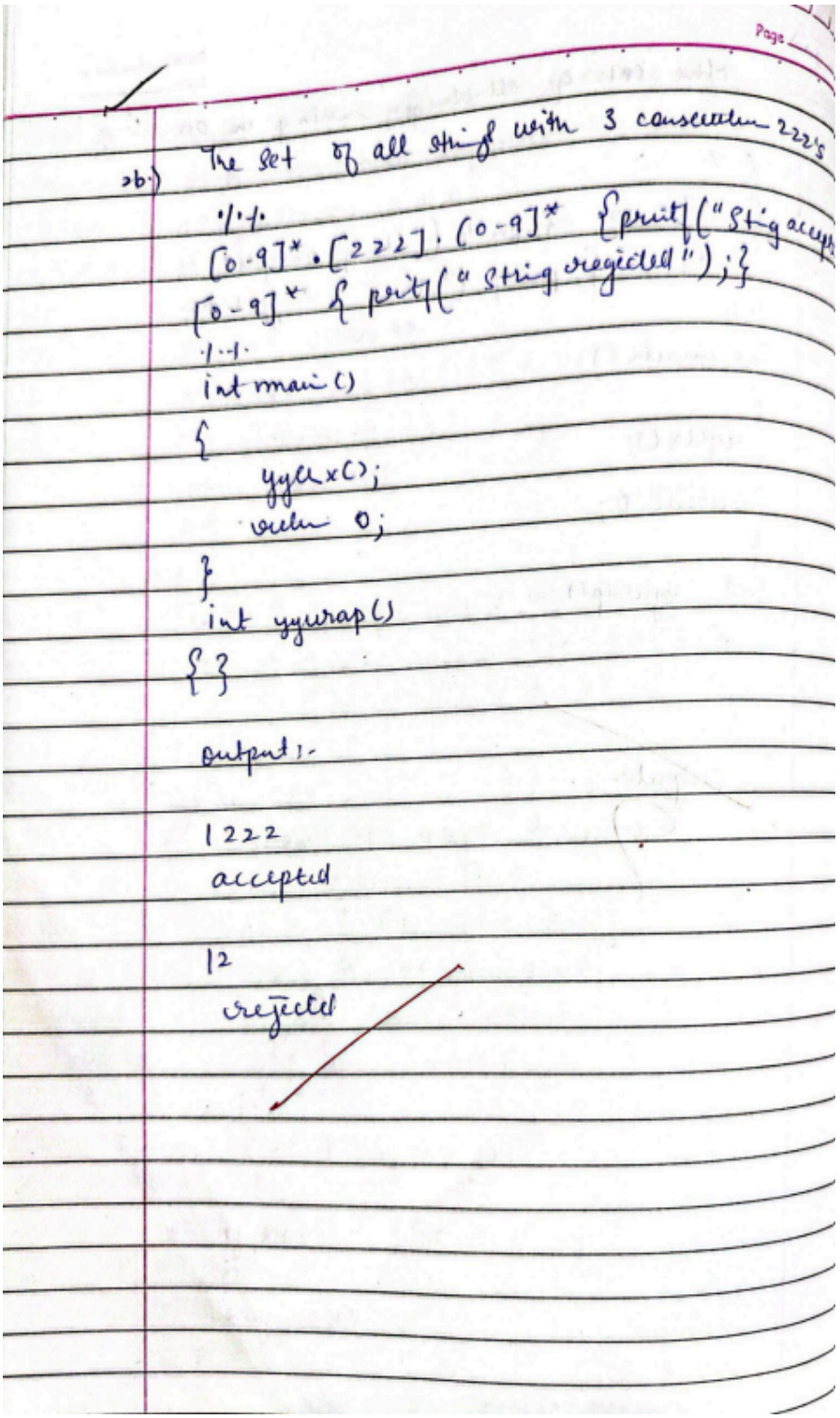

*text.txt
1 Hello World
2 Welcome to programming


Printed!

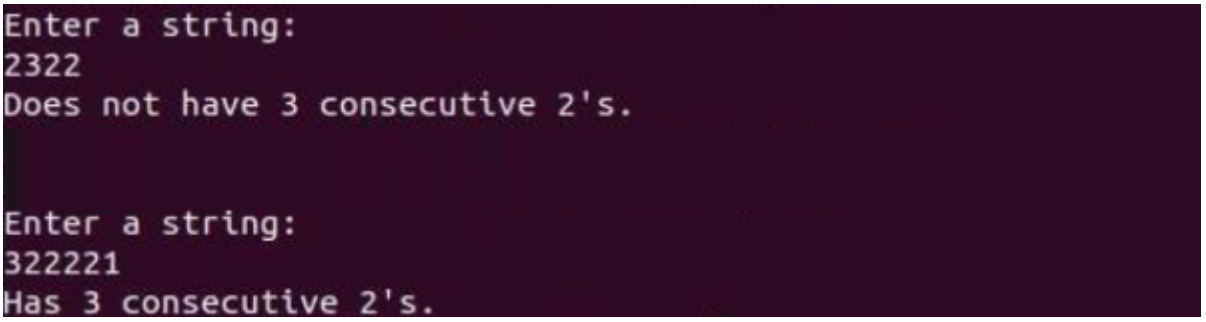

Open
print.txt
~/Documents
1 Hello World
2 Welcome to programming

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

Code

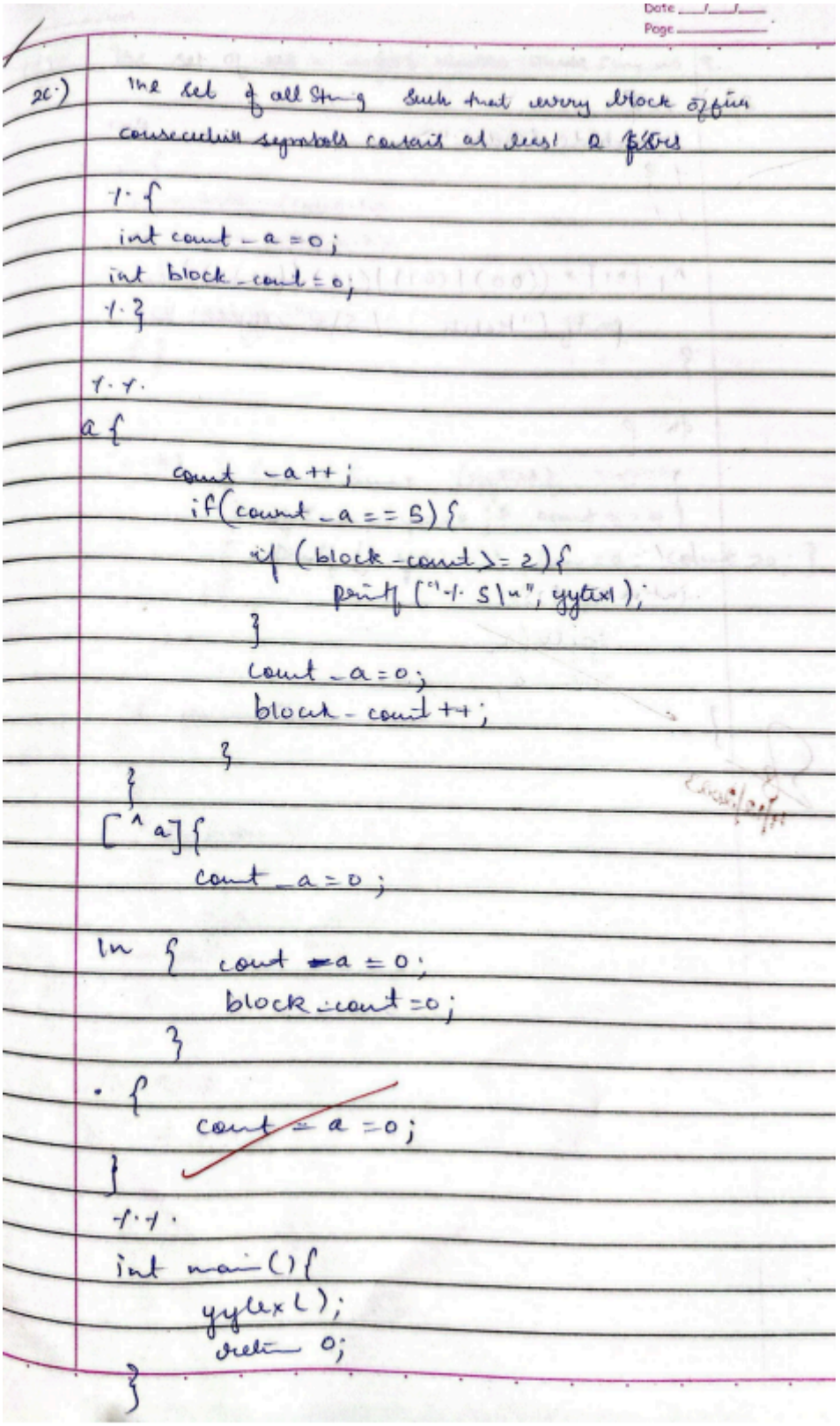


Output


Enter a string:
2322
Does not have 3 consecutive 2's.
Enter a string:
322221
Has 3 consecutive 2's.

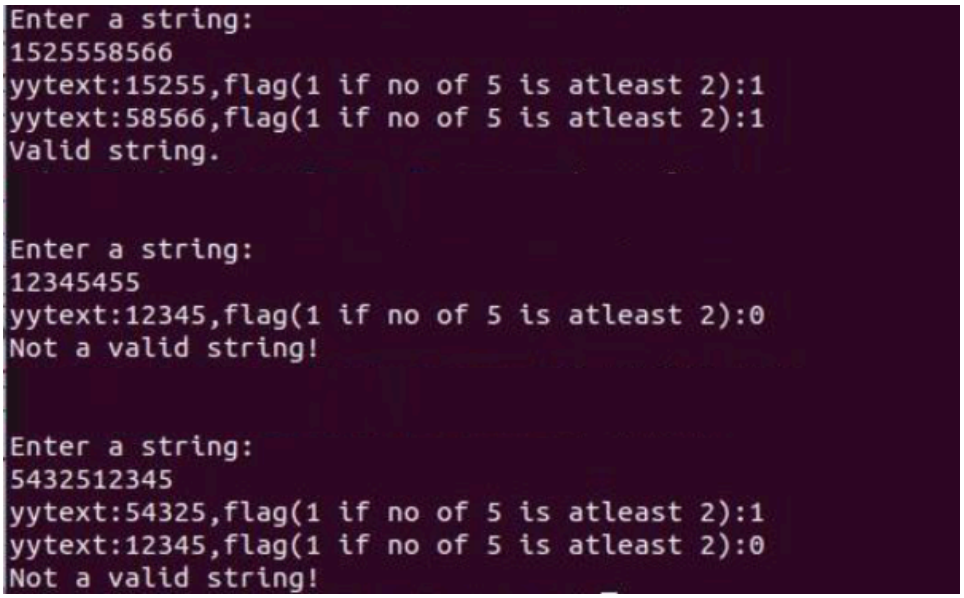
The set of all string such that every block of five consecutive symbols contains at least two 5's.

Code



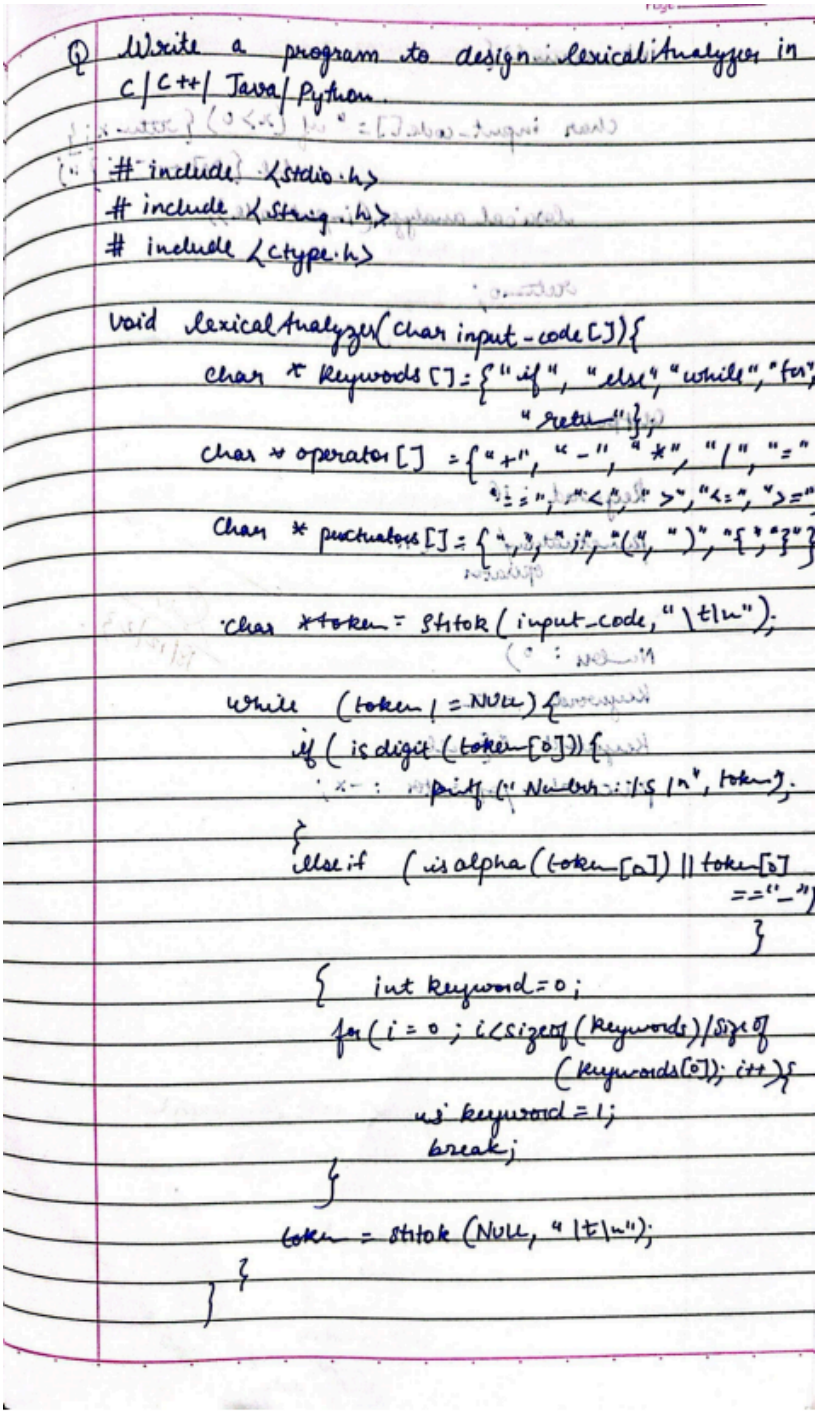
```
Date __/__/__
Page _____

2c.)  The set of all string such that every block of five
consecutive symbols contain at least 2 5's

%{
int count_a = 0;
int block_count = 0;
%}

%%
a{
    count_a++;
    if(count_a == 5){
        if (block_count >= 2){
            printf("%s\n", yytext);
        }
        count_a = 0;
        block_count++;
    }
}
[^a]{
    count_a = 0;
}
\n {  count_a = 0;
      block_count = 0;
   }
.{
    count_a = 0;
}
%%
int main(){
    yylex();
    return 0;
}
```

Output


Enter a string:
1525558566
yytext:15255,flag(1 if no of 5 is atleast 2):1
yytext:58566,flag(1 if no of 5 is atleast 2):1
Valid string.
Enter a string:
12345455
yytext:12345,flag(1 if no of 5 is atleast 2):0
Not a valid string!
Enter a string:
5432512345
yytext:54325,flag(1 if no of 5 is atleast 2):1
yytext:12345,flag(1 if no of 5 is atleast 2):0
Not a valid string!

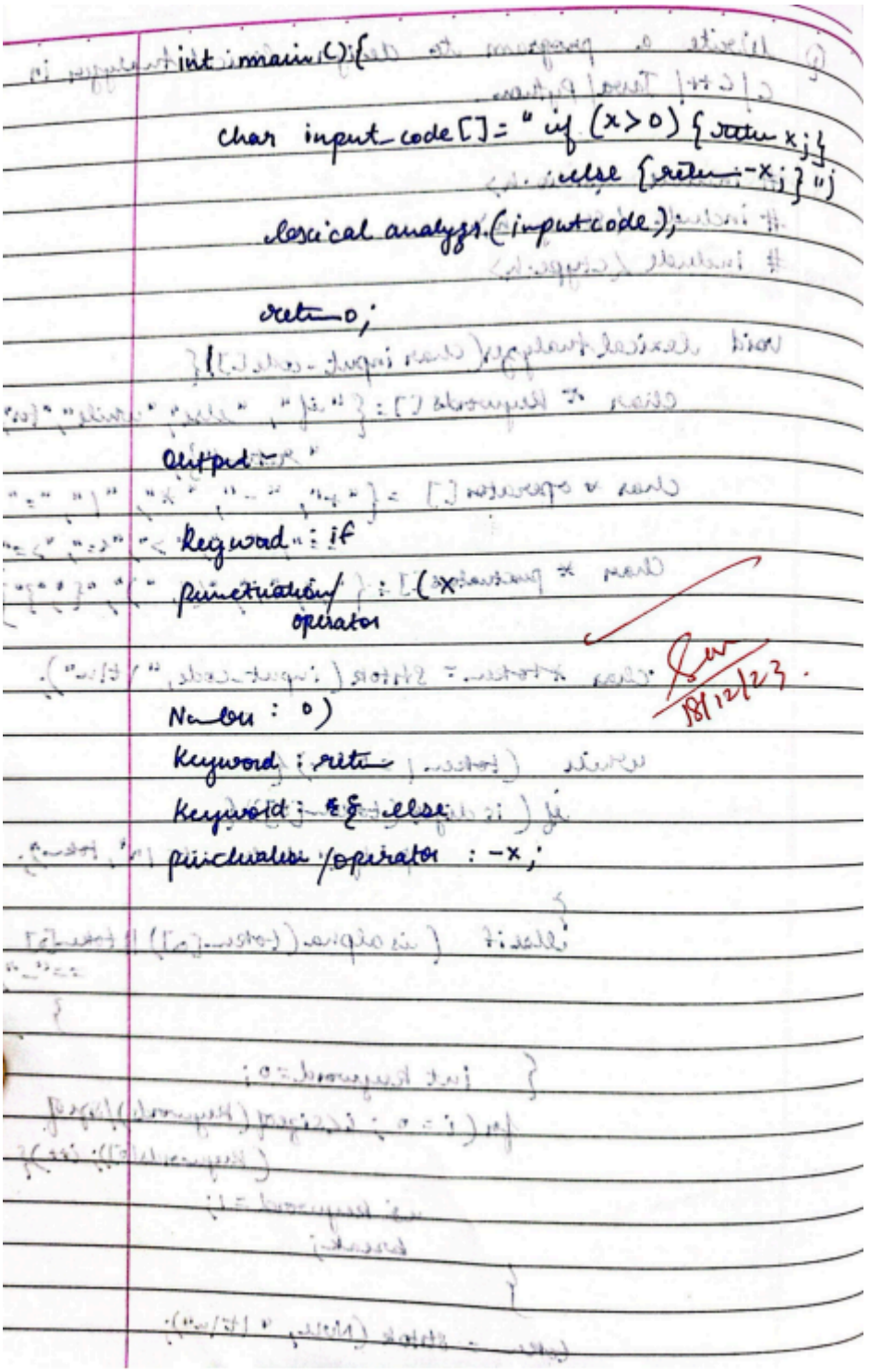
Write a C program to design lexical analysis to recognize any five keywords, identifiers, numbers, operators and punctuations.

Code



Q Write a program to design a lexical analyzer in C/C++/Java/Python.

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

void lexicalAnalyzer(char input_code[]){
    char * keywords[] = {"if", "else", "while", "for",
                         "return"};
    char * operator[] = {"+", "-", "*", "/", "=",
                         ":=", "<", ">", "<=", ">="};
    char * punctuators[] = {",", ";", "(", ")", "{", "}"};

    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 keyword = 0;
            for(i = 0; i < sizeof(keywords)/sizeof
                (keywords[0]); i++){
                keyword = 1;
                break;
            }
            token = strtok(NULL, " \t\n");
        }
    }
```


int main(){
char input_code[] = " if (x>0) { return x; }
else { return -x; } ";
lexical analyzer(input code);
return 0;
}
Output:-
" Keyword : if
punctuation/
operator
Number : 0)
Keyword : return
Keyword : else
punctuation/operator : -x;
}
18/12/23.

Output

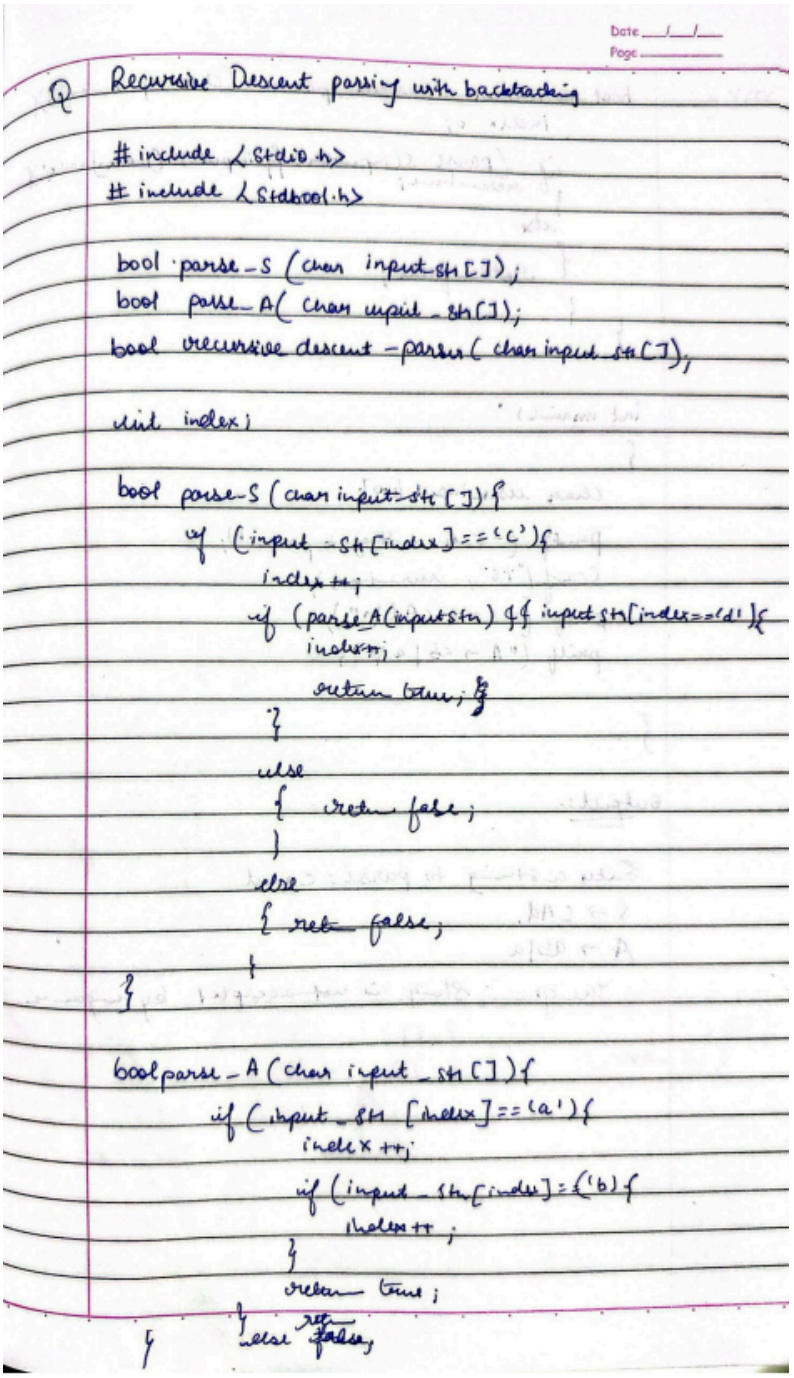
```
Keyword: if
Operator: (
Identifier: x
Operator: >
Number: 0
Operator: )
Operator: {
Keyword: return
Identifier: x
Punctuation:;
Operator: }
Keyword: else
Operator: {
Keyword: return
Operator: -x
Punctuation:;
Operator: }
```

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

S->cAd

A->ab | a

Code



```
bool recursive_descent_parser (char input_str[]){
    index = 0;
    if (parse_S(input_str) && input_str[index] == '$')
    {
        return true;
    }
    else
    {
        return false;
    }
}

int main()
{
    char user_input[100];
    printf("Enter a string to parse:");
    scanf("%s", user_input);
    printf("S → cAd\n");
    printf("A → ab|a\n");
}
```

Output:-

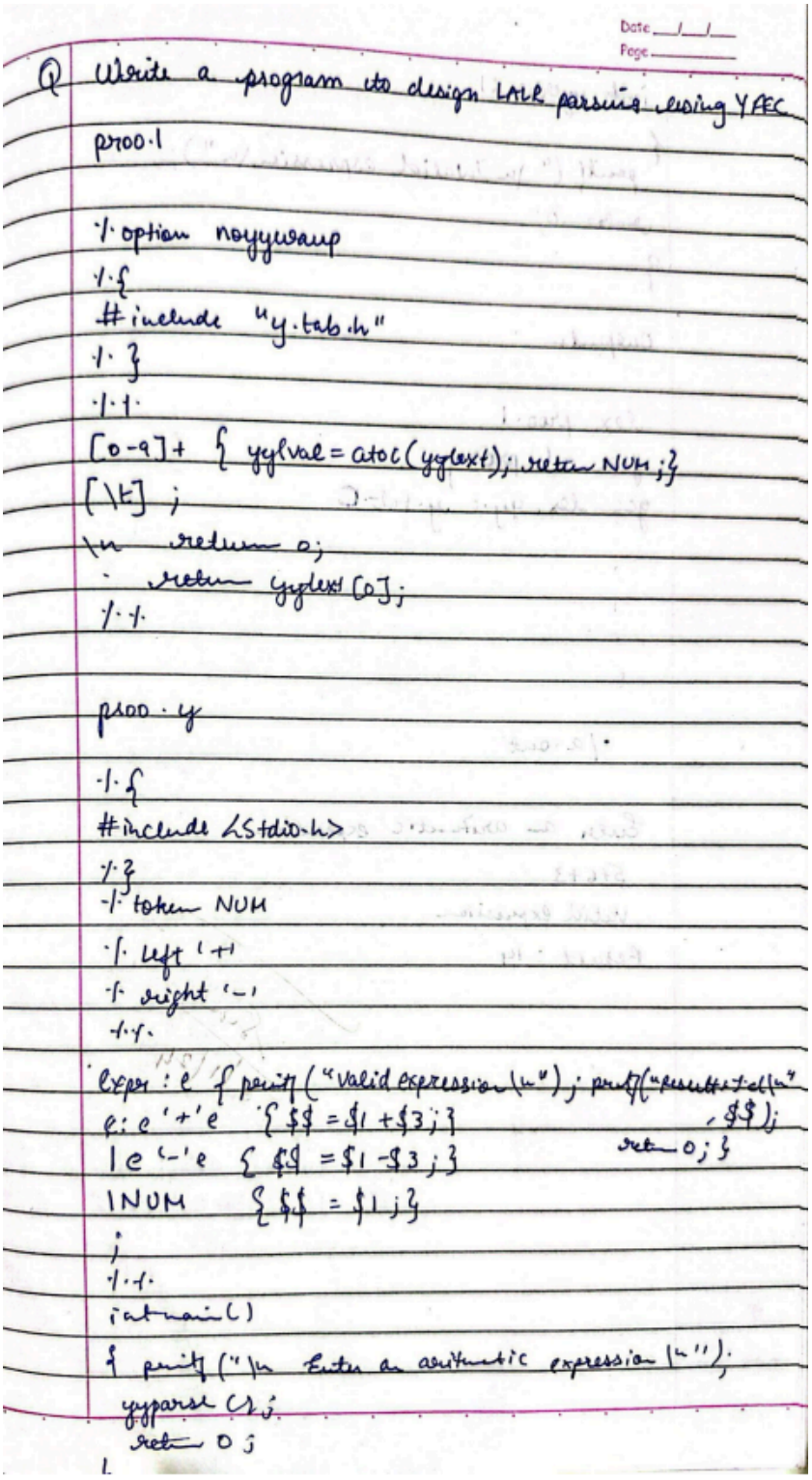
```
Enter a string to parse: caad
S → cAd
A → ab|a
The given string is not accepted by the grammar
```

Output

```
1.S->cAd
2.A->ab/a
Enter any string:cabd
String is accepted by the grammar
The productions used are
S -> cAd
A -> ab
```

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

Code



Q Write a program to design LALR parsing using YACC

proo.l

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

proo.y

```
%{
#include <stdio.h>
%}
%token NUM
%left '+'
%right '-'
%%
expr : e { printf("Valid expression\n"); printf("Result=%d\n", $$);
e: e '+' e { $$ = $1 + $3; }
| e '-' e { $$ = $1 - $3; }
| NUM { $$ = $1; }
;
%%
int main()
{ printf("\n Enter an arithmetic expression\n");
yyparse();
return 0;
}
```

```
int yyerror()
{
printf("\nInvalid expression\n");
return 0;
}
```

Output:

```
lex prog.l
yacc -d prog.y
gcc lex.yy.c y.tab.c
.
.
.
./a.out
```

Enter an arithmetic expression

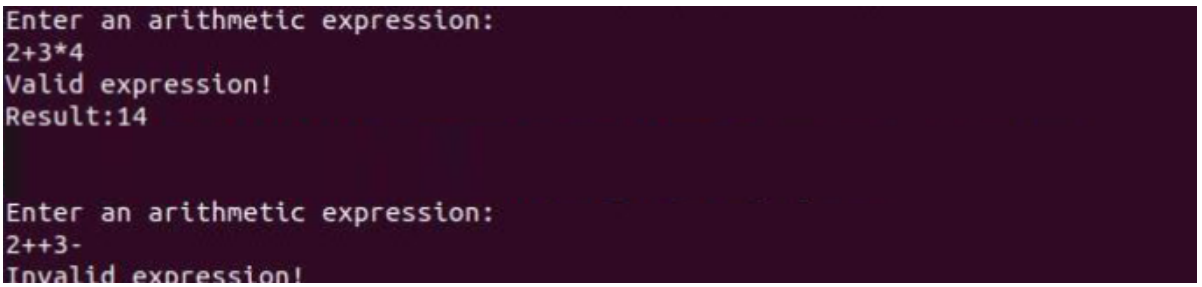
5+6+3

Valid expression

Result : 14

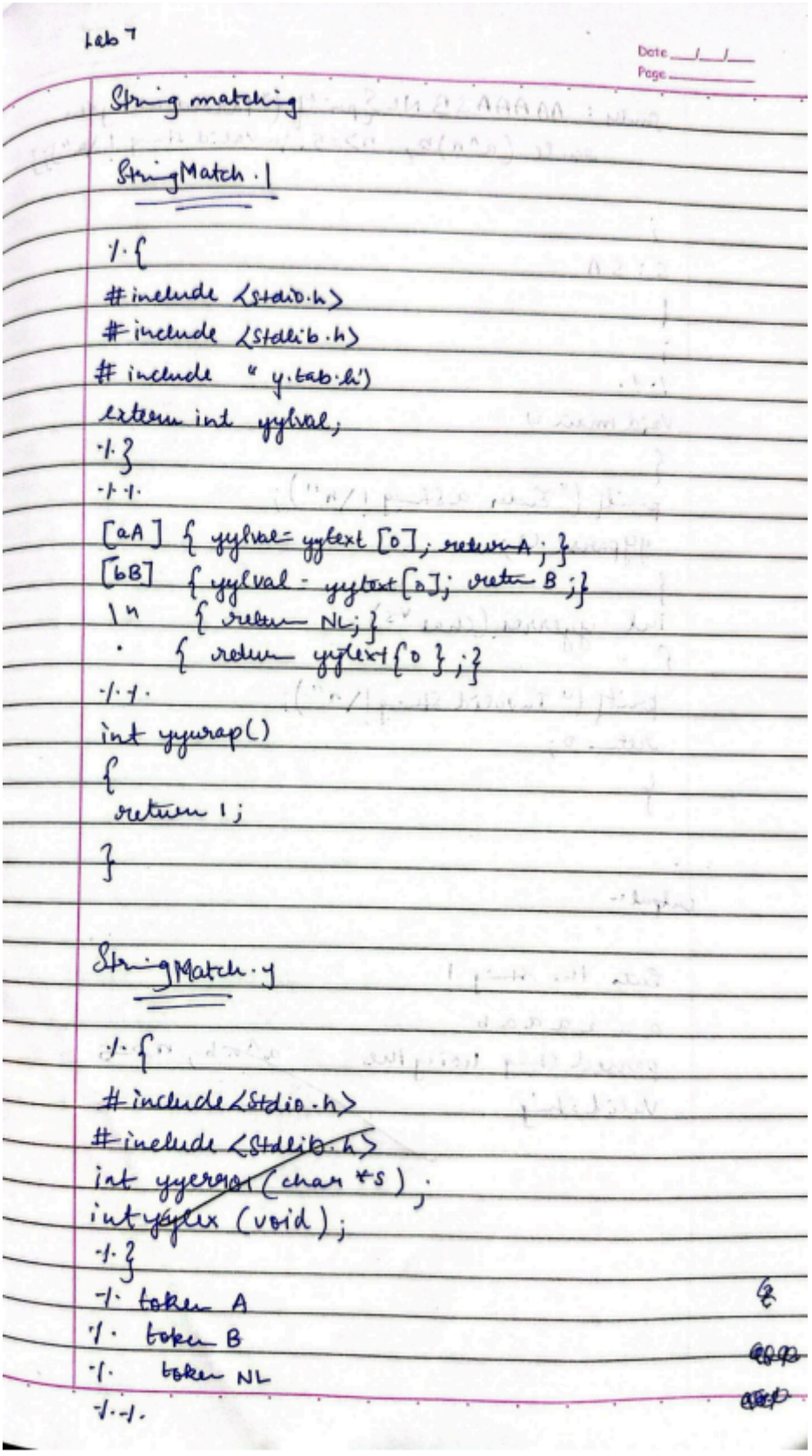
8/1/24

Output


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

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

Code



Lab 7

Date __/__/__
Page _____

String matching

StringMatch.l

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

StringMatch.y

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

Smdts : AAAAASBNL{printf("Parsed using the rule (a^n)b, n>=5.\n valid string !.\n");}

;

S : SA

|

;

%%

Void main()

{

printf("Enter a string !\n");

yyparse();

}

int yyerror(char *s)

{

printf("Invalid string !\n");

ret 0;

}

Output :-

Enter the string !

a a a a a a b

parsed string using the rule a^n b, n>=5

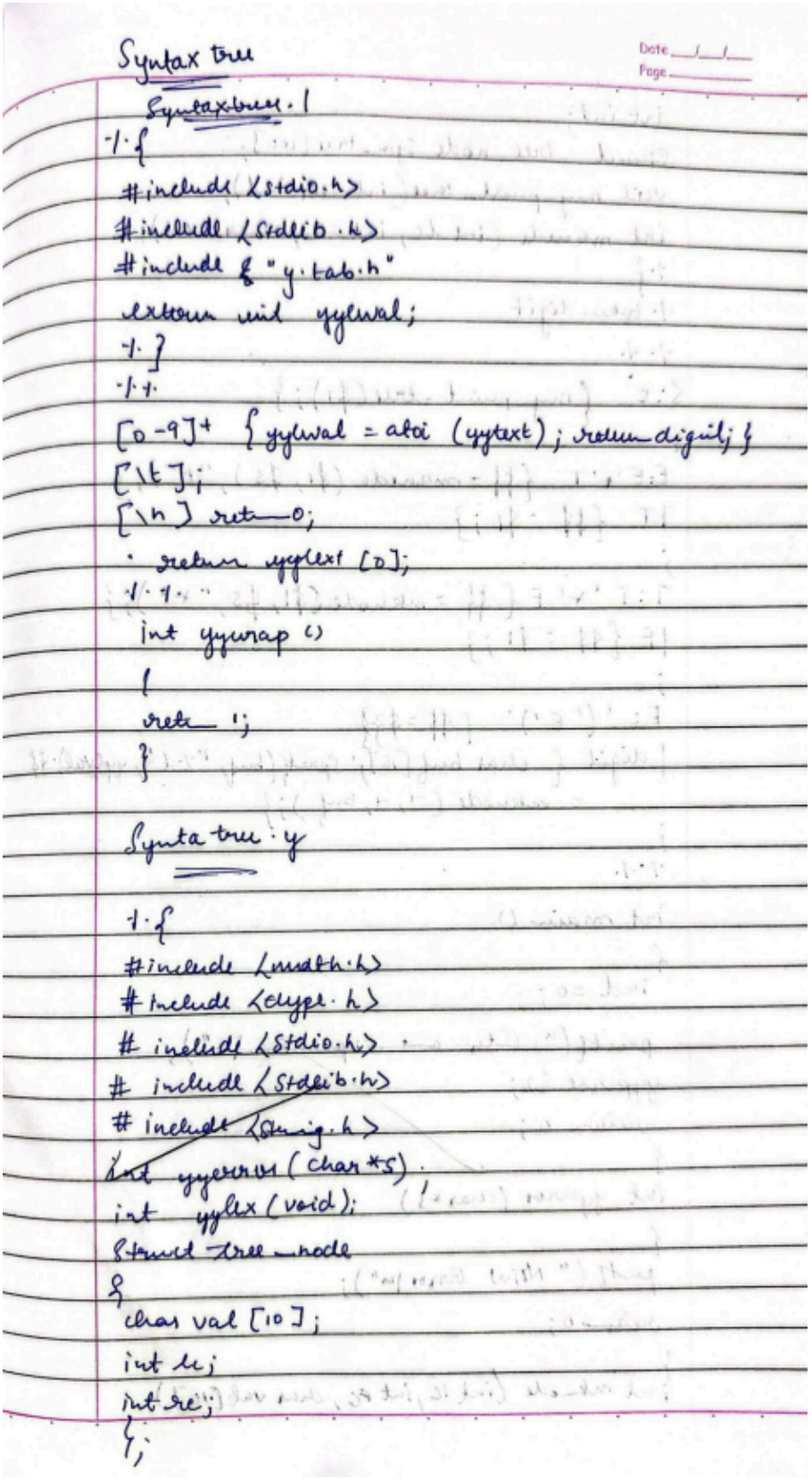
Valid string

Output


Enter a string!
aaaaaaab
Parsed using the rule (a^n)b, n>=5.
Valid String!
ab
Invalid String!
Enter a string!
abc
Invalid String!

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

Code



Syntax tree

Syntaxtree.l

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

Synta tree.y

```
%{
#include <math.h>
#include <ctype.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
void yyerror (char *s);
int yylex (void);
struct tree_node
{
char val[10];
int le;
int re;
};
```

```
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 {$$ = mknode ($1, $3, "+");}
| T {$$ = $1;}
;
T: T '*' F {$$ = mknode ($1, $3, "*");}
| F {$$ = $1;}
;
F: '(' E ')' {$$ = $2;}
| digit { char buf[10]; sprintf(buf, "%d", yylval); $$
    = mknode (-1, -1, buf);}
;
%%
int main ()
{
 ind = 0;
 printf("Enter an expression:\n");
 yyparse ();
 return 0;
}
int yyerror (char *s)
{
printf ("NITW Error\n");
return 0;
}
int mknode (int lc, int rc, char val[10])
```

```
{
strcpy( syn_tree[ind], val, val );
syn_tree[ind].lc = lc;
syn_tree[ind].rc = rc;
ind++;
return ind-1;
}
void my_print_tree(int cur_ind)
{
if( cur_ind == -1 ) return;
if ( syn_tree[cur_ind].lc == 1 && syn_tree[cur_ind].
                                    rc == -1)

my_print_tree(syn_tree[cur_ind].lc),
                                    rc)
}
```

Output :-

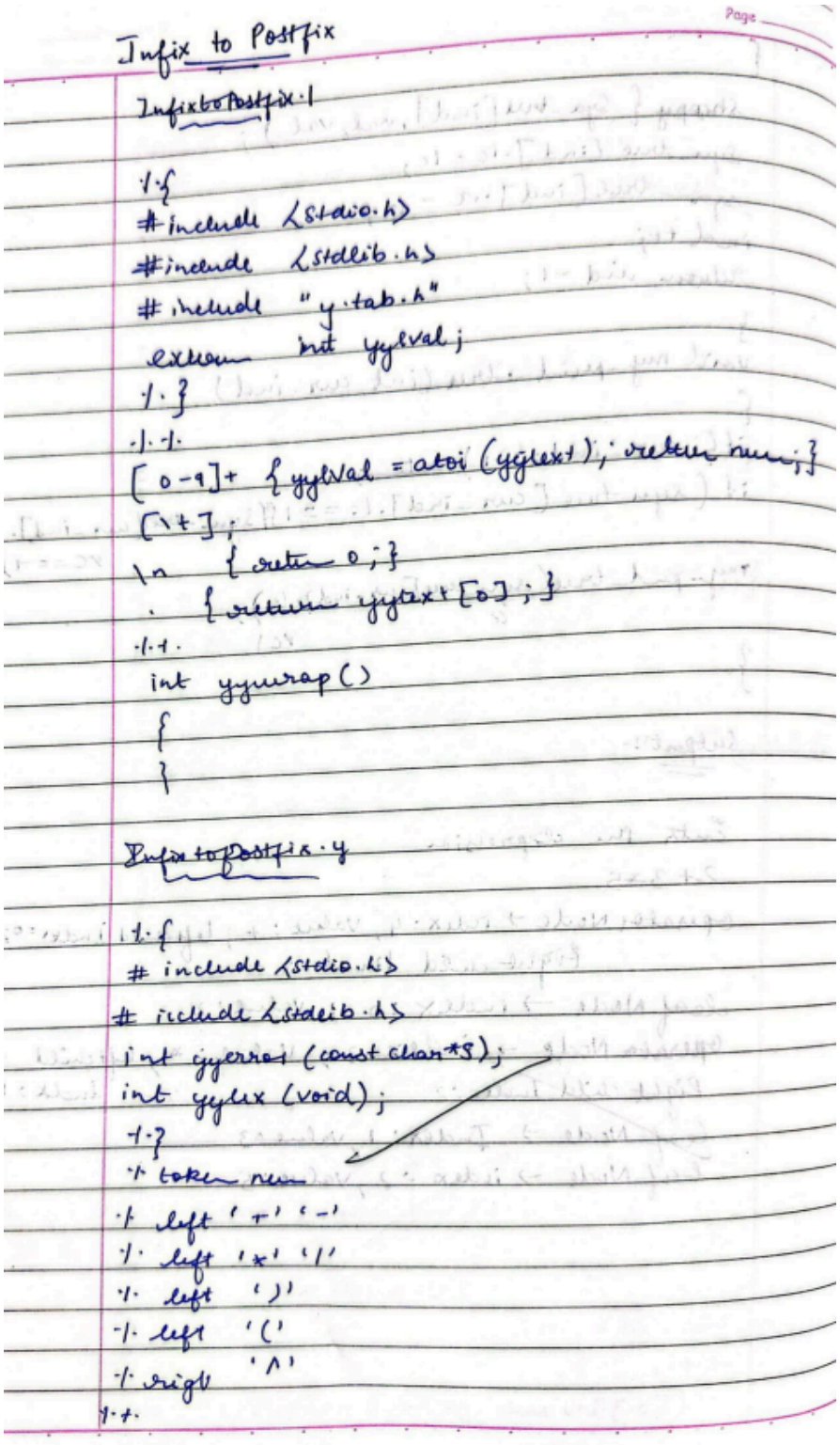
Enter the expression
2+3*5
Operator Node → index : 4, value : +, leftchild Index : 0;
Right child Index : 3
leaf Node → index : 0, value : 2
Operator Node → index : 3, value : *, leftchild Index : 1
Right child Index : 2
leaf Node → Index : 1, value : 3
leaf Node → index : 2, value : 5

Output

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

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

Code



Infix to Postfix

Infixtopostfix.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()
{
}
```

Infixtopostfix.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 { printf("/"); }
| h
;
h : f '^' h { printf("^"); }
| f
;
f : '(' 'e' ')'
| num { printf("%d", $1); }
;
%%

Void main()
{
printf("Enter a infix exp : \n");
yyparse();
}
int yyerror(const char *s)
{
printf("Invalid infix exp \n");
return 0;
}
```

output:-

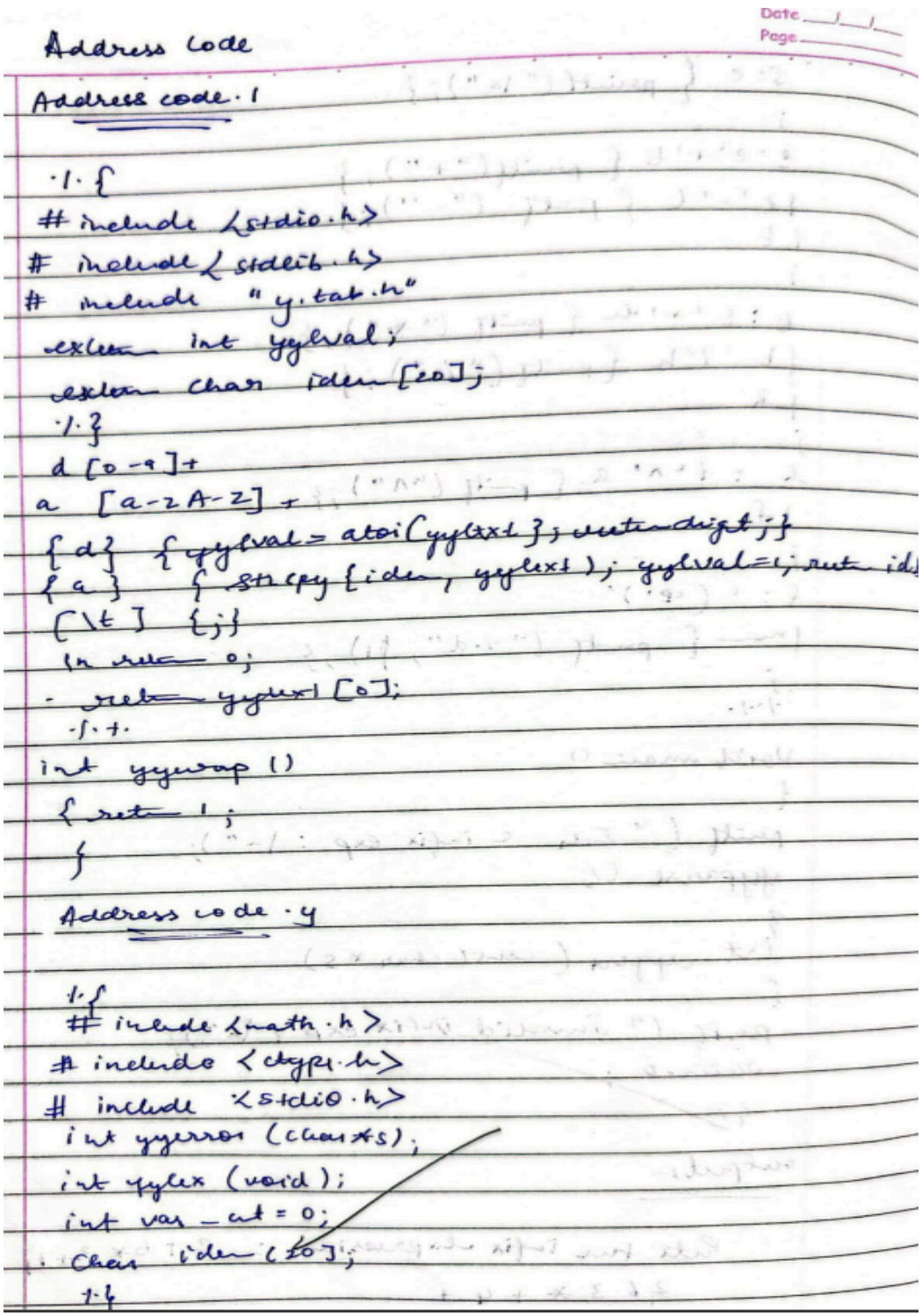
Enter the infix expression : 2+6*3+4

2 6 3 * + 4 +

Output

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

Code



Address code

Address code.l

```
%{
#include <stdio.h>
#include <stdlib.h>
#include "y.tab.h"
extern int yylval;
extern char iden[20];
%}
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;
}
```

Address code.y

```
%{
#include <math.h>
#include <ctype.h>
#include <stdio.h>
int yyerror(char *s);
int yylex(void);
int var_cnt = 0;
char iden[20];
%}
```

```
%{
%token id
%token digit
%%
S: id '=' E {printf("%d = t%d\n", id, var_cnt-1);}
E: 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;}
;
F: '(' E ')' {$$ = $2;}
| id {$$ = id;}
| digit {$$ = digit;}
;
%%
int main()
{
var_cnt = 0;
printf("Enter an expression:\n");
yyparse();
return 0;
}
int yyerror(char *s)
{
printf("Invalid expression\n");
return 0;
}
```

output:

```
Enter an exp:
a = 2 + 3 * 6
t0 = 2;
t1 = 3;
t2 = 6;
t3 = t1 * t2
t4 = 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
```