VISVESVARAYA TECHNOLOGICAL UNIVERSITY

“JnanaSangama”, Belgaum -590014, Karnataka.

LAB REPORT

On

COMPILER DESIGN

Submitted by

HARSHITHA R (1BM21CS075)

Under the Guidance of

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

Nov-2023 to Feb-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 **HARSHITHA R (1BM21CS075)** 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 Visveswaraya Technological University, Belgaum during the year 2023-2024. The Lab report has been approved as it satisfies the academic requirements in respect of **Compiler Design (22CS5PCCPD)** work prescribed for the said degree.

Signature of the Guide	**Signature of the HOD**
Prameetha Pai	Dr. Jyothi S Nayak
Assistant Professor	Prof.& Head, Dept. of CSE
BMSCE, Bengaluru	BMSCE, Bengaluru

TABLE OF CONTENTS

Sl. No	TITLE	Pg. No
1	Write a program in LEX to identify keywords, identifiers and separators in a C program.	1
2	Write a program in LEX to identify whether the entered input is a number, operator or invalid character. It should ignore whitespace and tab space.	2
3	Write a program in LEX to ECHO	3
4	Write a program in LEX to identify data type- int, char, float and variable.	4
5	Write a program in LEX to identify each character as vowels or consonants.	5
6	Write a program in LEX to identify alphabets as characters and numbers as digits.	6
7	Write a program in LEX to count the number of words in an input sentence.	7
8	Write a program in LEX to count the number of vowels and consonants in a given string.	8
9	Write a program in LEX to identify alphanumeric strings.	9
10	Read input from file and print on the terminal	10
11	Read input from a file and the output should be stored in another file.	11-12
12	Write a Lex program to read and input sentences, and check if it is compound or simple.	13
13	Write a LEX program to read an input sentence and to check if the sentence begins with English articles.	14
14	Write a program to check if the input sentence ends with any of the following punctuation marks.	15

15	Write a program to read and check if the user entered number is signed or unsigned using appropriate meta character.	16
16	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	17
17	Write a program in LEX to recognize Floating Point Numbers.	18
18	Write a LEX program that copies a file, replacing each nonempty sequence of white spaces by a single blank.	19
19	Write a LEX program to recognize the following tokens over the alphabets {0,1,..,9}	20-29
	a) The set of all string ending in 00.	20
	b) The set of all strings with three consecutive 222's.	21
	c) The set of all string such that every block of five consecutive symbols contains at least two 5's.	22-23
	d) The set of all strings beginning with a 1 which, interpreted as the binary representation of an integer, is congruent to zero modulo 5.	24-25
	e) The set of all strings such that the 10th symbol from the right end is 1.	26
	f) The set of all four digits numbers whose sum is 9.	27-28
	g) The set of all four digital numbers, whose individual digits are in ascending order from left to right.	29
20	Write a program to design Lexical Analyzer to recognize any five keywords, identifiers, numbers, operators and punctuations.	30-32
21	Write a Program to perform Recursive Descent Parsing on the following grammar. S->cAd , A->ab /a	33-34
22	Write a YACC program to implement desk calculator	35-36

23	Write a YACC program to generate a syntax tree for a given arithmetic expression.	37-39
24	Use YACC to convert: Infix expression to Postfix expression.	40-41
25	Write a YACC program to recognize the grammar (a^nb, n>=5)	42-43
26	Use YACC to generate 3-Address code for a given expression	44-46

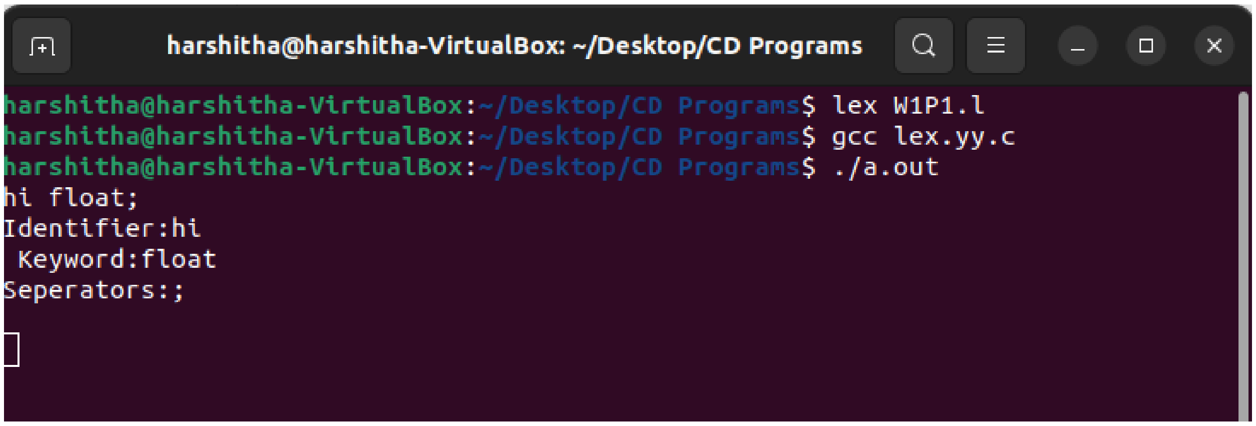
PROGRAM 1

Write a program in LEX to identify keywords, identifiers and separators in a C program.

Code:

```
%option noyywrap
%{
  #include<stdio.h>
%}
%%
int|float|char {printf("Keyword:%s \n",yytext);}
[a-zA-Z][a-zA-Z0-9] {printf("Identifier:%s \n",yytext);}
,|; {printf("Seperators:%s \n",yytext);}
%%
void main()
{
  yylex();
}
```

Output:



```
harshitha@harshitha-VirtualBox:~/Desktop/CD Programs$ lex W1P1.l
harshitha@harshitha-VirtualBox:~/Desktop/CD Programs$ gcc lex.yy.c
harshitha@harshitha-VirtualBox:~/Desktop/CD Programs$ ./a.out
hi float;
Identifier:hi
 Keyword:float
Seperators:;
```

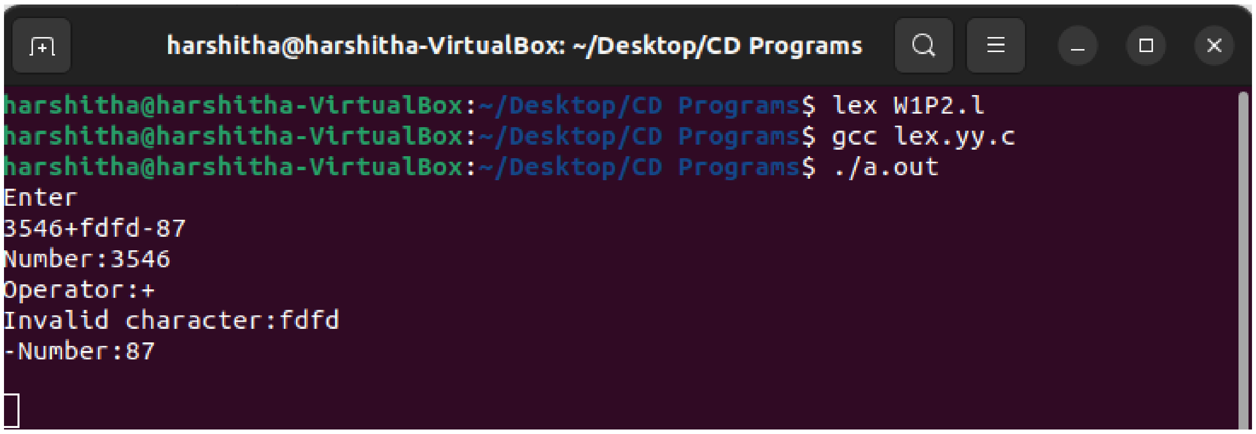
PROGRAM 2

Write a program in LEX to identify whether the entered input is a number, operator or invalid character. It should ignore whitespace and tab space.

Code:

```
%option noyywrap
%{
  #include<stdio.h>
%}
%%
[0-9]+ {printf("Number:%s \n",yytext);}
[+|-|*|/] {printf("Operator:%s \n",yytext);}
[ \t\n] {/*ignore whitespace and newline */}
[a-zA-Z]+ {printf("Invalid character:%s \n",yytext);}
%%
void main()
{
  printf("Enter \n");
  yylex();
}
```

Output:



```
harshitha@harshitha-VirtualBox:~/Desktop/CD Programs$ lex W1P2.l
harshitha@harshitha-VirtualBox:~/Desktop/CD Programs$ gcc lex.yy.c
harshitha@harshitha-VirtualBox:~/Desktop/CD Programs$ ./a.out
Enter
3546+fdfd-87
Number:3546
Operator:+
Invalid character:fdfd
-Number:87

```

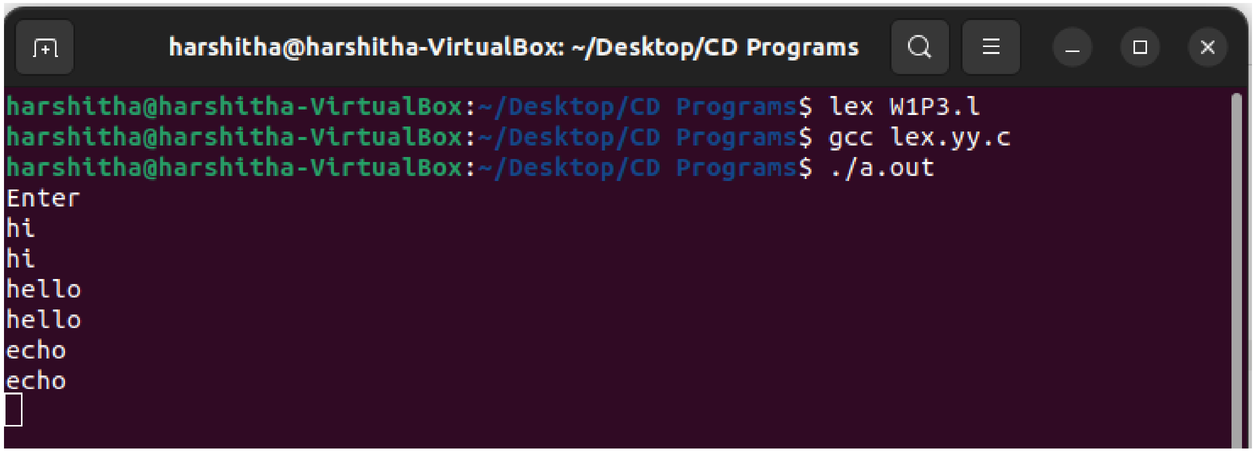
PROGRAM 3

Write a program in LEX to ECHO

Code:

```
%option noyywrap
%%
.ECHO
%%
void main()
{
 printf("Enter \n");
 yylex();
}
```

Output:



```
harshitha@harshitha-VirtualBox: ~/Desktop/CD Programs
harshitha@harshitha-VirtualBox:~/Desktop/CD Programs$ lex W1P3.l
harshitha@harshitha-VirtualBox:~/Desktop/CD Programs$ gcc lex.yy.c
harshitha@harshitha-VirtualBox:~/Desktop/CD Programs$ ./a.out
Enter
hi
hi
hello
hello
echo
echo
```

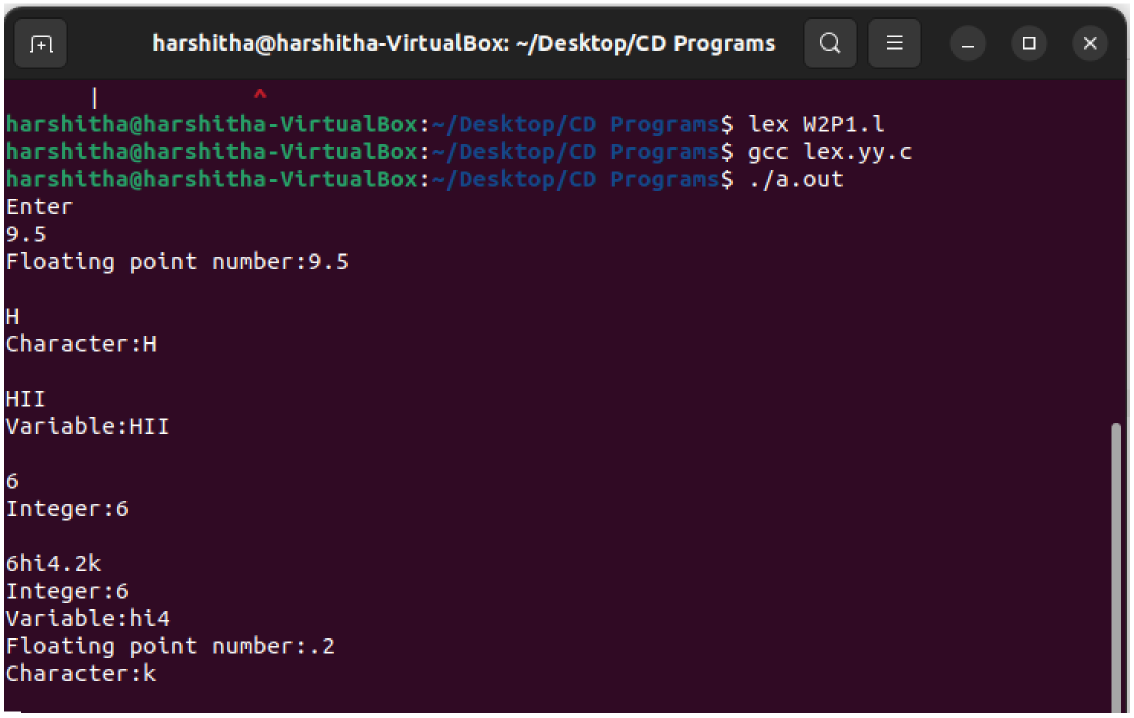
PROGRAM 4

Write a program in LEX to identify data type- int, char, float and variable.

Code:

```
%option noyywrap
%{
  #include<stdio.h>
%}
%%
[+-]?[0-9]+ {printf("Integer:%s \n",yytext);}
[a-zA-Z] {printf("Character:%s \n",yytext);}
[+-]?[0-9]*[.][0-9]+ {printf("Floating point number:%s \n",yytext);}
[a-zA-Z]+[a-zA-Z0-9]+ {printf("Variable:%s \n",yytext);}
%%
void main()
{
 printf("Enter\n");
 yylex();
}
```

Output:



```
harshitha@harshitha-VirtualBox:~/Desktop/CD Programs$ lex W2P1.l
harshitha@harshitha-VirtualBox:~/Desktop/CD Programs$ gcc lex.yy.c
harshitha@harshitha-VirtualBox:~/Desktop/CD Programs$ ./a.out
Enter
9.5
Floating point number:9.5

H
Character:H

HII
Variable:HII

6
Integer:6

6hi4.2k
Integer:6
Variable:hi4
Floating point number:.2
Character:k
```

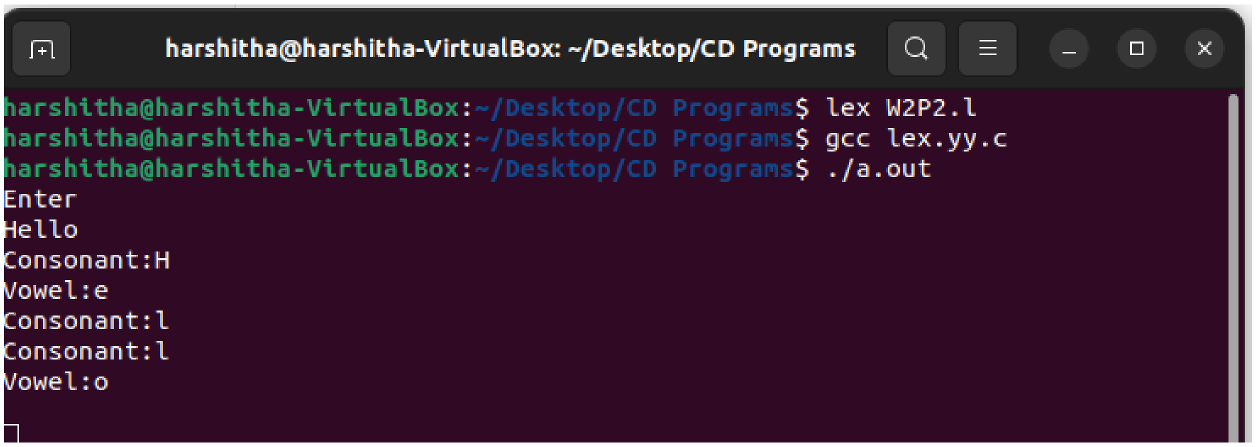
PROGRAM 5

Write a program in LEX to identify each character as vowels or consonants.

Code:

```
%option noyywrap
%{
  #include<stdio.h>
%}
%%
[aeiouAEIOU] {printf("Vowel:%s \n",yytext);}
[a-zA-Z] {printf("Consonant:%s \n",yytext);}
%%
void main()
{
 printf("Enter\n");
 yylex();
}
```

Output:



```
harshitha@harshitha-VirtualBox:~/Desktop/CD Programs$ lex W2P2.l
harshitha@harshitha-VirtualBox:~/Desktop/CD Programs$ gcc lex.yy.c
harshitha@harshitha-VirtualBox:~/Desktop/CD Programs$ ./a.out
Enter
Hello
Consonant:H
Vowel:e
Consonant:l
Consonant:l
Vowel:o
```

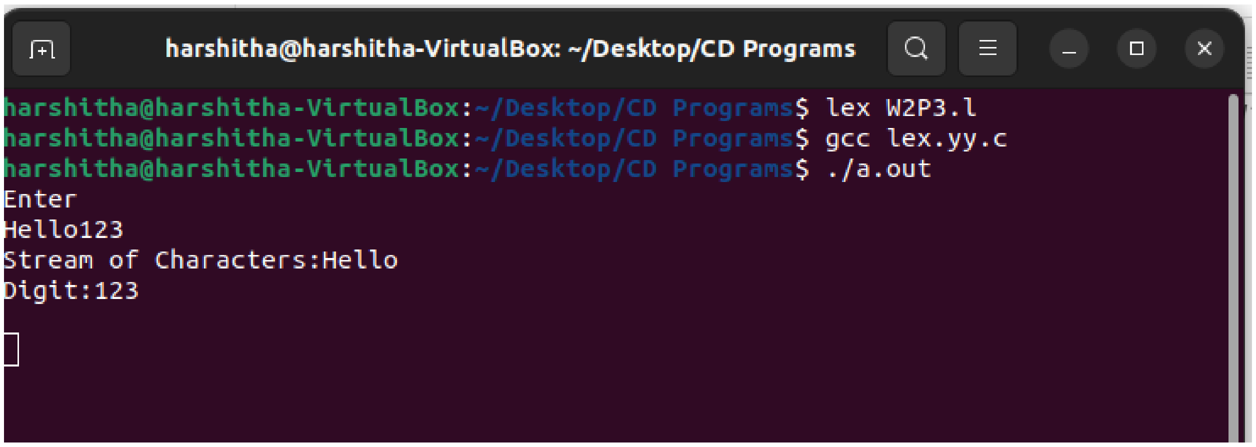
PROGRAM 6

Write a program in LEX to identify alphabets as characters and numbers as digits.

Code:

```
%option noyywrap
%{
  #include<stdio.h>
%}
%%
[+-]?[0-9]+ {printf("Digit:%s \n",yytext);}
[a-zA-Z]+ {printf("Stream of Characters:%s \n",yytext);}
%%
void main()
{
 printf("Enter\n");
 yylex();
}
```

Output:



```
harshitha@harshitha-VirtualBox:~/Desktop/CD Programs$ lex W2P3.l
harshitha@harshitha-VirtualBox:~/Desktop/CD Programs$ gcc lex.yy.c
harshitha@harshitha-VirtualBox:~/Desktop/CD Programs$ ./a.out
Enter
Hello123
Stream of Characters:Hello
Digit:123
```

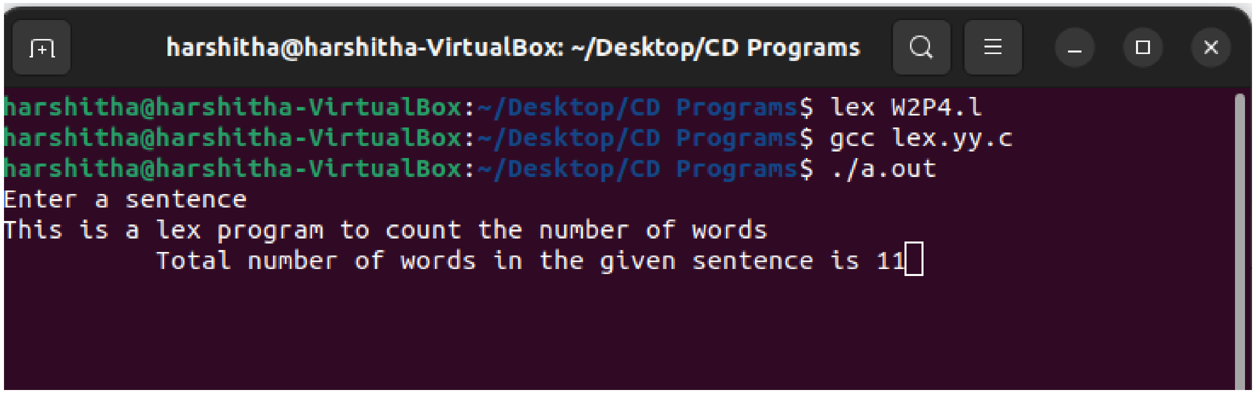
PROGRAM 7

Write a program in LEX to count the number of words in an input sentence.

Code:

```
%option noyywrap
%{
  #include<stdio.h>
%}
 int len=0;
%%
[a-zA-Z0-9]+ {len++;};
"\n" {printf("Total number of words in the given sentence is %d",len);}
%%
void main()
{
 printf("Enter a sentence\n");
 yylex();
}
```

Output:



```
harshitha@harshitha-VirtualBox:~/Desktop/CD Programs$ lex W2P4.l
harshitha@harshitha-VirtualBox:~/Desktop/CD Programs$ gcc lex.yy.c
harshitha@harshitha-VirtualBox:~/Desktop/CD Programs$ ./a.out
Enter a sentence
This is a lex program to count the number of words
          Total number of words in the given sentence is 11
```

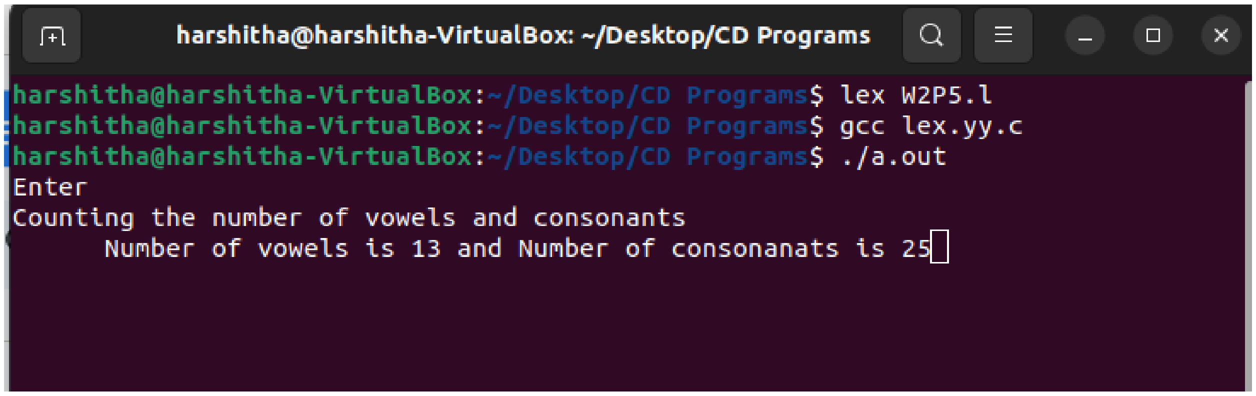
PROGRAM 8

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

Code:

```
%option noyywrap
%{
  #include<stdio.h>
%}
 int len=0;
%%
[a-zA-Z0-9]+ {len++;};
"\n" {printf("Total number of words in the given sentence is %d",len);}
%%
void main()
{
 printf("Enter a sentence\n");
 yylex();
}
```

Output:



```
harshitha@harshitha-VirtualBox:~/Desktop/CD Programs$ lex W2P5.l
harshitha@harshitha-VirtualBox:~/Desktop/CD Programs$ gcc lex.yy.c
harshitha@harshitha-VirtualBox:~/Desktop/CD Programs$ ./a.out
Enter
Counting the number of vowels and consonants
      Number of vowels is 13 and Number of consonanats is 25
```

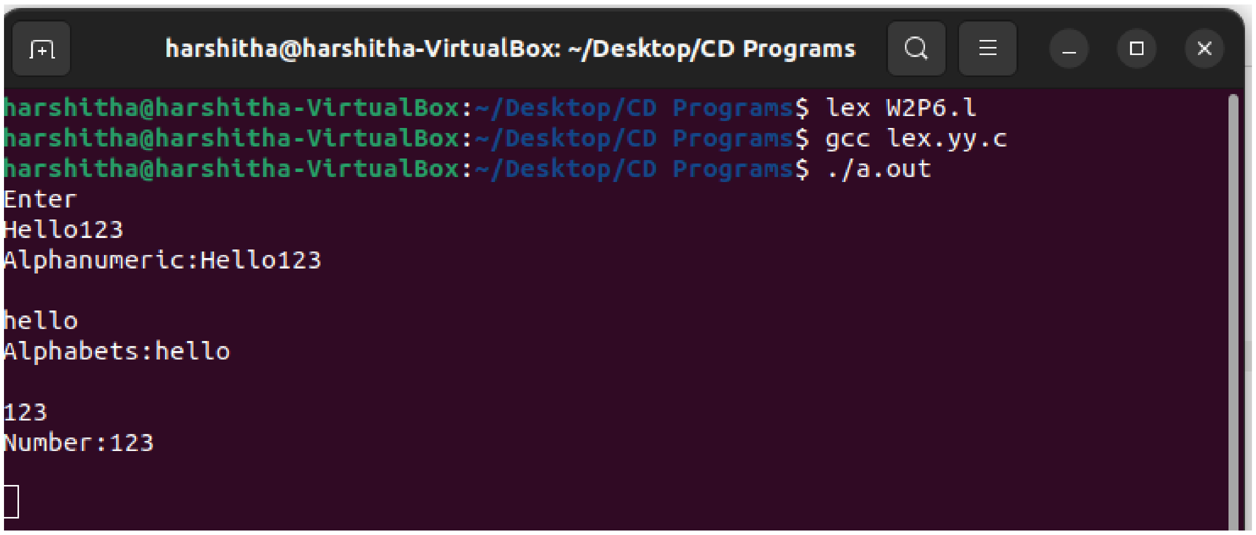
PROGRAM 9

Write a program in LEX to identify alphanumeric strings.

Code:

```
%option noyywrap
%{
  #include<stdio.h>
%}
%%
[+-]?[0-9]+ {printf("Number:%s \n",yytext);}
[a-zA-Z]+ {printf("Alphabets:%s \n",yytext);}
[a-zA-Z0-9]+ {printf("Alphanumeric:%s \n",yytext);}
%%
void main()
{
 printf("Enter\n");
 yylex();
}
```

Output:



```
harshitha@harshitha-VirtualBox:~/Desktop/CD Programs$ lex W2P6.l
harshitha@harshitha-VirtualBox:~/Desktop/CD Programs$ gcc lex.yy.c
harshitha@harshitha-VirtualBox:~/Desktop/CD Programs$ ./a.out
Enter
Hello123
Alphanumeric:Hello123

hello
Alphabets:hello

123
Number:123

```

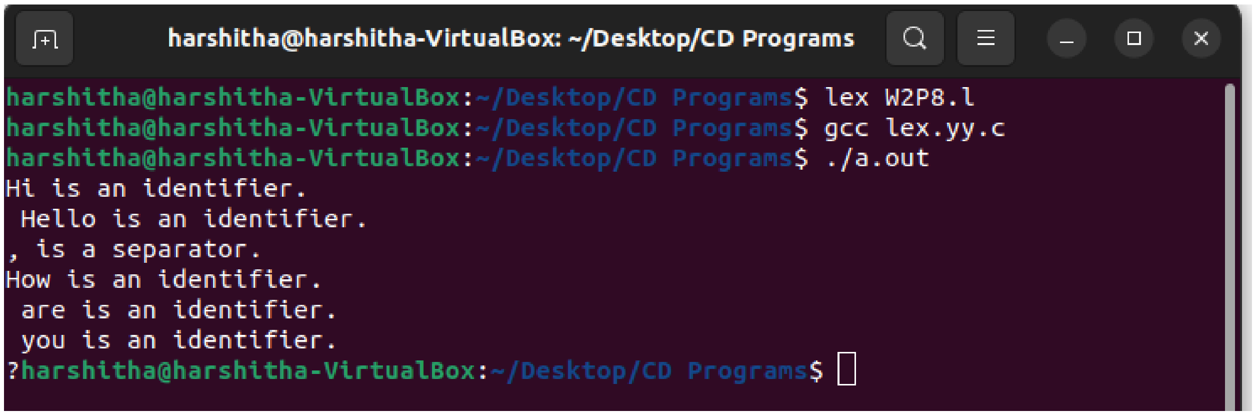
PROGRAM 10

Read input from file and print on the terminal

Code:

```
%option noyywrap
%{
#include<stdio.h>
%}
%%
char|int|float {printf("%s is a keyword.\n",yytext);}
[a-zA-Z][a-zA-Z0-9]* {printf("%s is an identifier.\n",yytext);}
, {printf("%s is a separator.\n",yytext);}
; {printf("%s is a delimiter.\n",yytext);}
"=" {printf("%s is an assignment operator.\n",yytext);}
"+"|"-"|"*"|"/" {printf("%s is a binary operator.\n",yytext);}
[0-9]+ {printf("%s is/are digit(s).\n",yytext);}
\n ;
%%

void main()
{
yyin=fopen("input.txt","r");
yylex();
fclose(yyin);
}
```

Output:



```
harshitha@harshitha-VirtualBox:~/Desktop/CD Programs$ lex W2P8.l
harshitha@harshitha-VirtualBox:~/Desktop/CD Programs$ gcc lex.yy.c
harshitha@harshitha-VirtualBox:~/Desktop/CD Programs$ ./a.out
Hi is an identifier.
 Hello is an identifier.
, is a separator.
How is an identifier.
 are is an identifier.
 you is an identifier.
?harshitha@harshitha-VirtualBox:~/Desktop/CD Programs$ 
```

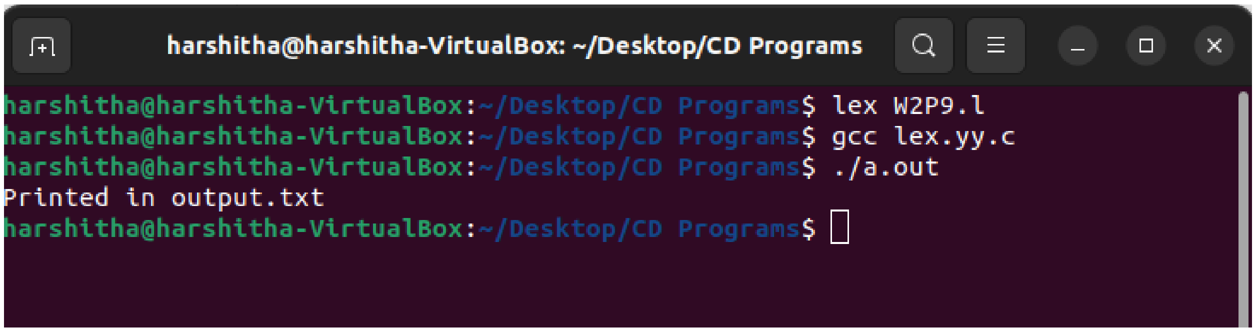
PROGRAM 11

Read input from a file and the output should be stored in another file.

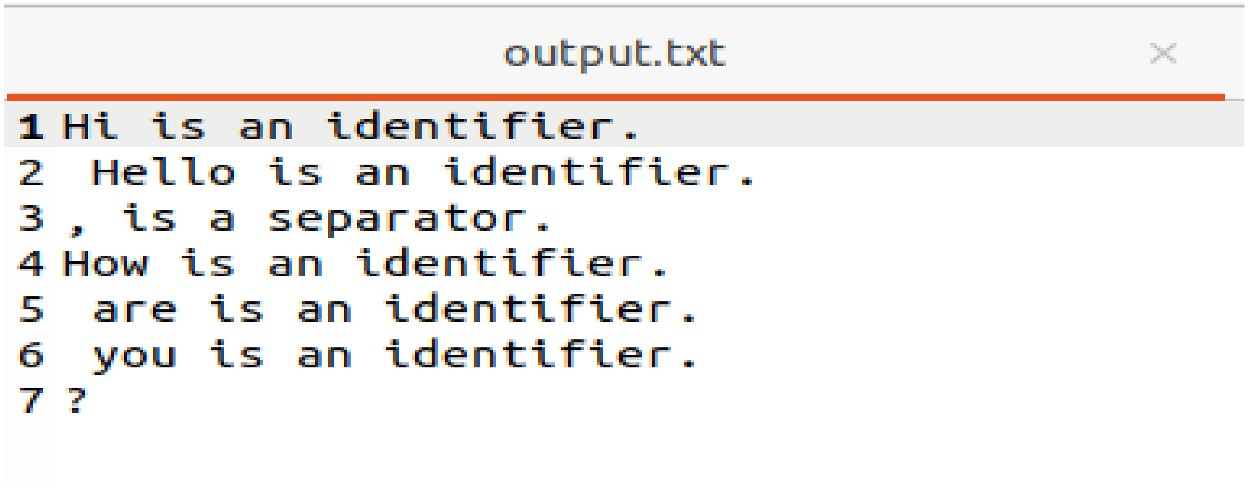
Code:

```
%{
#include<stdio.h>
%}
%%
char|int|float {fprintf(yyout,"%s is a keyword.\n",yytext);}
[a-zA-Z][a-zA-Z0-9]* {fprintf(yyout,"%s is an identifier.\n",yytext);}
, {fprintf(yyout,"%s is a separator.\n",yytext);}
; {fprintf(yyout,"%s is a delimiter.\n",yytext);}
"=" {fprintf(yyout,"%s is an assignment operator.\n",yytext);}
"+"|"-"|"*"|"/" {fprintf(yyout,"%s is a binary operator.\n",yytext);}
[0-9]+ {fprintf(yyout,"%s is/are digit(s).\n",yytext);}
\n ;
%%
void main()
{
yyin=fopen("input.txt","r");
yyout=fopen("output.txt","w");
yylex();
printf("Printed in output.txt\n");
fclose(yyin);
fclose(yyout);
}
int yywrap()
{
return 1;
}
```

Output:



```
harshitha@harshitha-VirtualBox:~/Desktop/CD Programs$ lex W2P9.l
harshitha@harshitha-VirtualBox:~/Desktop/CD Programs$ gcc lex.yy.c
harshitha@harshitha-VirtualBox:~/Desktop/CD Programs$ ./a.out
Printed in output.txt
harshitha@harshitha-VirtualBox:~/Desktop/CD Programs$ 
```



output.txt

```
Hi is an identifier.
 Hello is an identifier.
, is a separator.
How is an identifier.
 are is an identifier.
 you is an identifier.
?
```

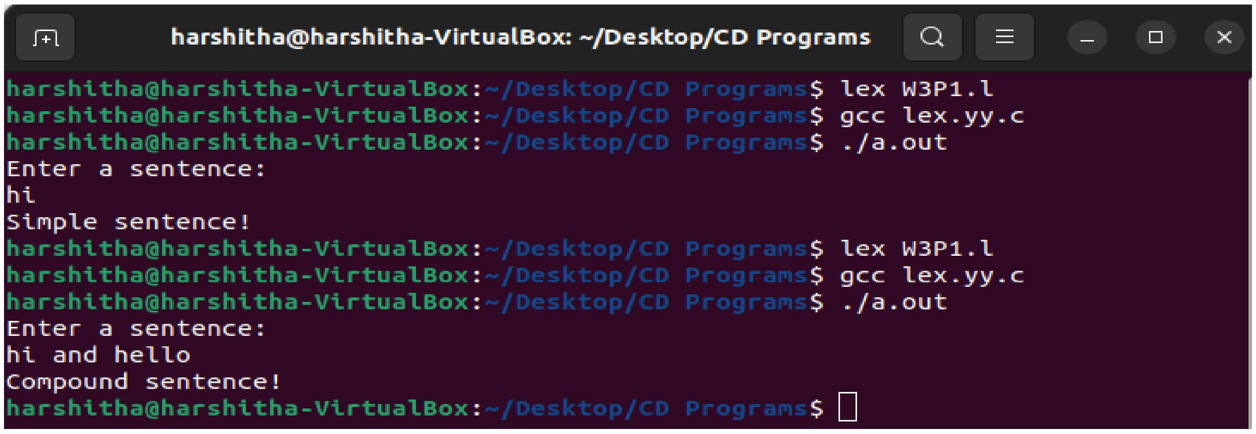
PROGRAM 12

Write a Lex program to read and input sentences, 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:

```
%option noyywrap
%{
#include<stdio.h>
int flag=0;
%}
%%
if|then|but|because|nevertheless|and|or {flag=1;}
.* {flag=0;}
\n {return 0;}
%%
void main()
{
printf("Enter a sentence:\n");
yylex();
if(flag==1)
printf("Compound sentence!\n");
else
printf("Simple sentence!\n");
}
```

Output:



harshitha@harshitha-VirtualBox: ~/Desktop/CD Programs

```
harshitha@harshitha-VirtualBox:~/Desktop/CD Programs$ lex W3P1.l
harshitha@harshitha-VirtualBox:~/Desktop/CD Programs$ gcc lex.yy.c
harshitha@harshitha-VirtualBox:~/Desktop/CD Programs$ ./a.out
Enter a sentence:
hi
Simple sentence!
harshitha@harshitha-VirtualBox:~/Desktop/CD Programs$ lex W3P1.l
harshitha@harshitha-VirtualBox:~/Desktop/CD Programs$ gcc lex.yy.c
harshitha@harshitha-VirtualBox:~/Desktop/CD Programs$ ./a.out
Enter a sentence:
hi and hello
Compound sentence!
harshitha@harshitha-VirtualBox:~/Desktop/CD Programs$
```

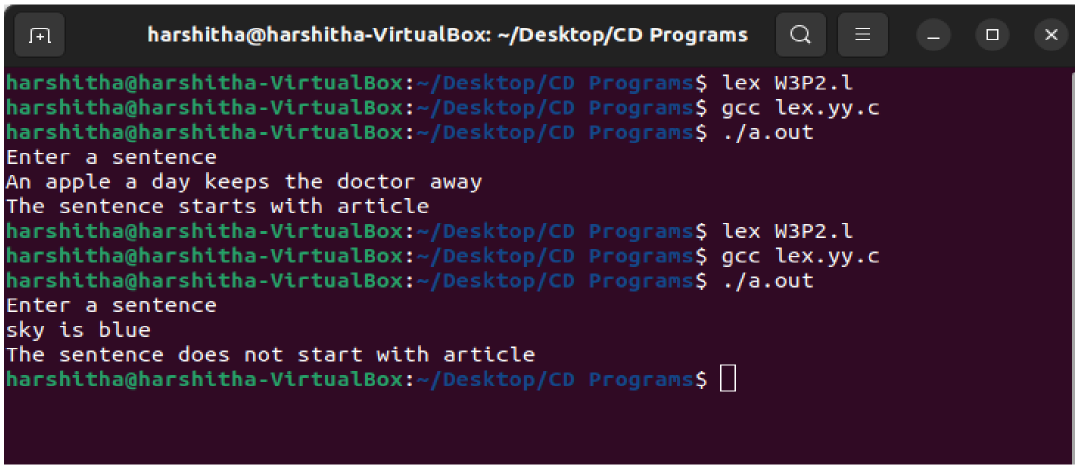
PROGRAM 13

Write a LEX 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

Code:

```
%option noyywrap
%{
  #include<stdio.h>
  int flag=0;
%}
%%
^(a|an|the|A|An|The)[" "].* {flag=1;}
.* {flag=0;}
\n {return 0;}
%%
void main()
{
 printf("Enter a sentence\n");
 yylex();
 if(flag==1)
  printf("The sentence starts with article\n");
 else
  printf("The sentence does not start with article\n");
}
```

Output:



```
harshitha@harshitha-VirtualBox:~/Desktop/CD Programs$ lex W3P2.l
harshitha@harshitha-VirtualBox:~/Desktop/CD Programs$ gcc lex.yy.c
harshitha@harshitha-VirtualBox:~/Desktop/CD Programs$ ./a.out
Enter a sentence
An apple a day keeps the doctor away
The sentence starts with article
harshitha@harshitha-VirtualBox:~/Desktop/CD Programs$ lex W3P2.l
harshitha@harshitha-VirtualBox:~/Desktop/CD Programs$ gcc lex.yy.c
harshitha@harshitha-VirtualBox:~/Desktop/CD Programs$ ./a.out
Enter a sentence
sky is blue
The sentence does not start with article
harshitha@harshitha-VirtualBox:~/Desktop/CD Programs$
```

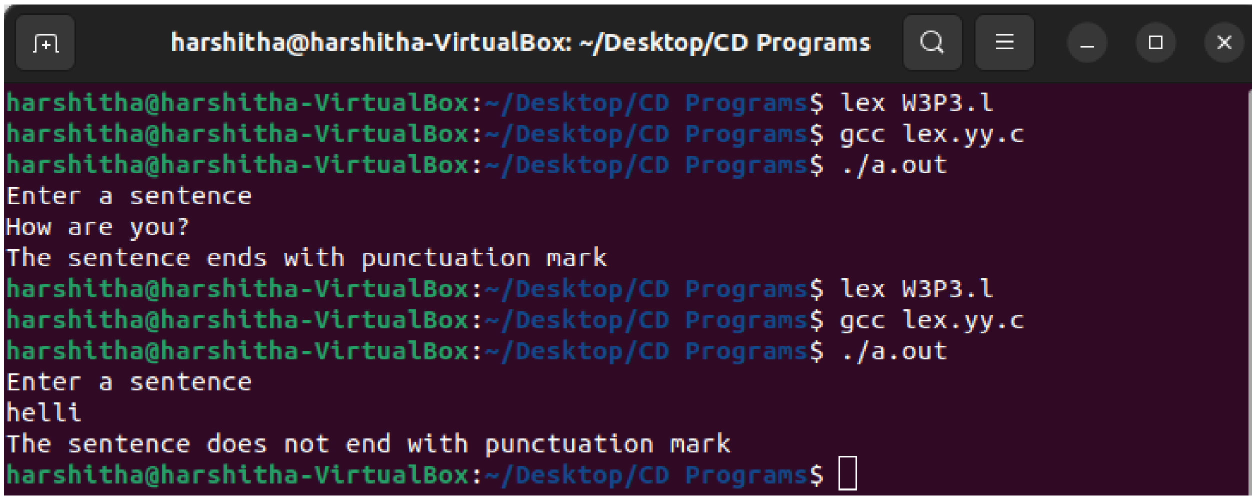
PROGRAM 14

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

Code:

```
%option noyywrap
%{
  #include<stdio.h>
  int flag=0;
%}
%%
.*[?|!|.]$ {flag=1;}
.* {flag=0;}
\n {return 0;}
%%
void main()
{
 printf("Enter a sentence\n");
 yylex();
 if(flag==1)
  printf("The sentence ends with punctuation mark\n");
 else
  printf("The sentence does not end with punctuation mark\n");
}
```

Output:



harshitha@harshitha-VirtualBox: ~/Desktop/CD Programs

```
harshitha@harshitha-VirtualBox:~/Desktop/CD Programs$ lex W3P3.l
harshitha@harshitha-VirtualBox:~/Desktop/CD Programs$ gcc lex.yy.c
harshitha@harshitha-VirtualBox:~/Desktop/CD Programs$ ./a.out
Enter a sentence
How are you?
The sentence ends with punctuation mark
harshitha@harshitha-VirtualBox:~/Desktop/CD Programs$ lex W3P3.l
harshitha@harshitha-VirtualBox:~/Desktop/CD Programs$ gcc lex.yy.c
harshitha@harshitha-VirtualBox:~/Desktop/CD Programs$ ./a.out
Enter a sentence
helli
The sentence does not end with punctuation mark
harshitha@harshitha-VirtualBox:~/Desktop/CD Programs$ 
```

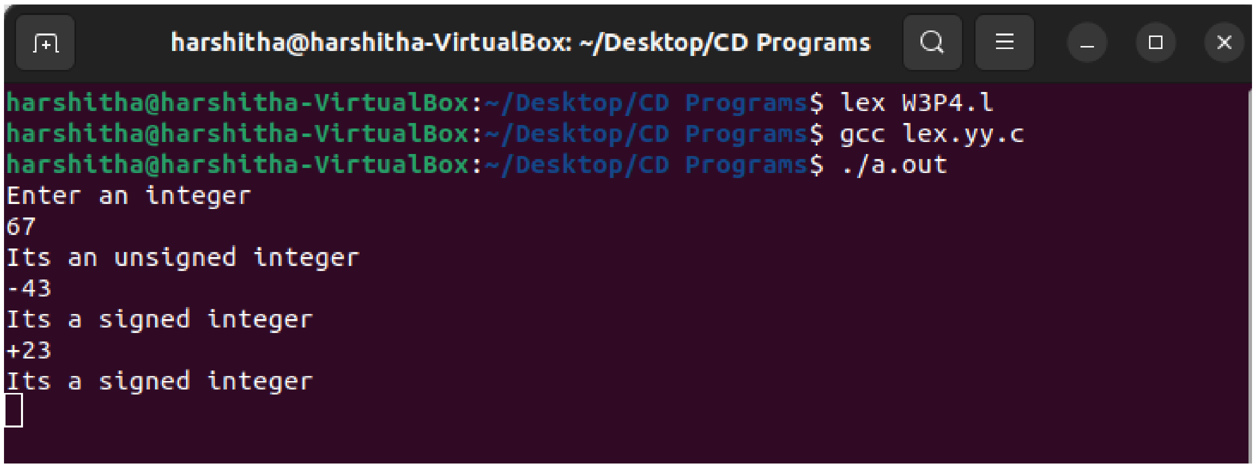
PROGRAM 15

Write a program to read and check if the user entered number is signed or unsigned using appropriate meta character

Code:

```
%option noyywrap
%{
  #include<stdio.h>
%}
%%
[+-][0-9]+ {printf("Its a signed integer");}
[0-9]+ {printf("Its an unsigned integer");}
%%
void main()
{
 printf("Enter an integer\n");
 yylex();
}
```

Output:



```
harshitha@harshitha-VirtualBox:~/Desktop/CD Programs$ lex W3P4.l
harshitha@harshitha-VirtualBox:~/Desktop/CD Programs$ gcc lex.yy.c
harshitha@harshitha-VirtualBox:~/Desktop/CD Programs$ ./a.out
Enter an integer
67
Its an unsigned integer
-43
Its a signed integer
+23
Its a signed integer
```

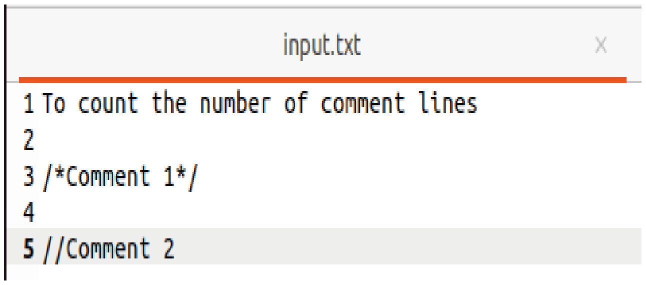
PROGRAM 16

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:

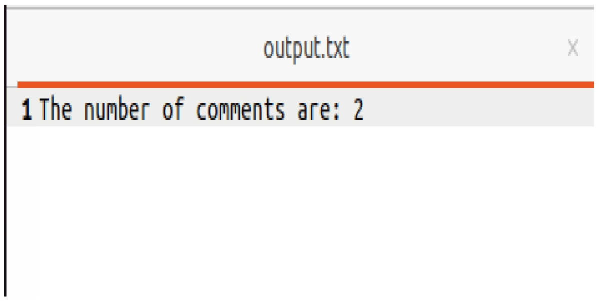
```
%{
#include<stdio.h>
int c=0;
%}
%%
"\/\*"[^*]*\*+([^/*][^*]*\*+)*\/ {c++;}
"//".* {c++;}
. ECHO;
%%
int yywrap()
{
return 1;
}
void main()
{
yyin=fopen("input.txt","r");
yyout=fopen("output.txt","w");
yylex();
printf("The number of comments are:%d\n",c);
fprintf(yyout, "The number of comments are: %d\n", c);
fclose(yyin);
fclose(yyout);
}
```

Output:



input.txt

```
To count the number of comment lines

/*Comment 1*/

//Comment 2
```



output.txt

```
The number of comments are: 2
```

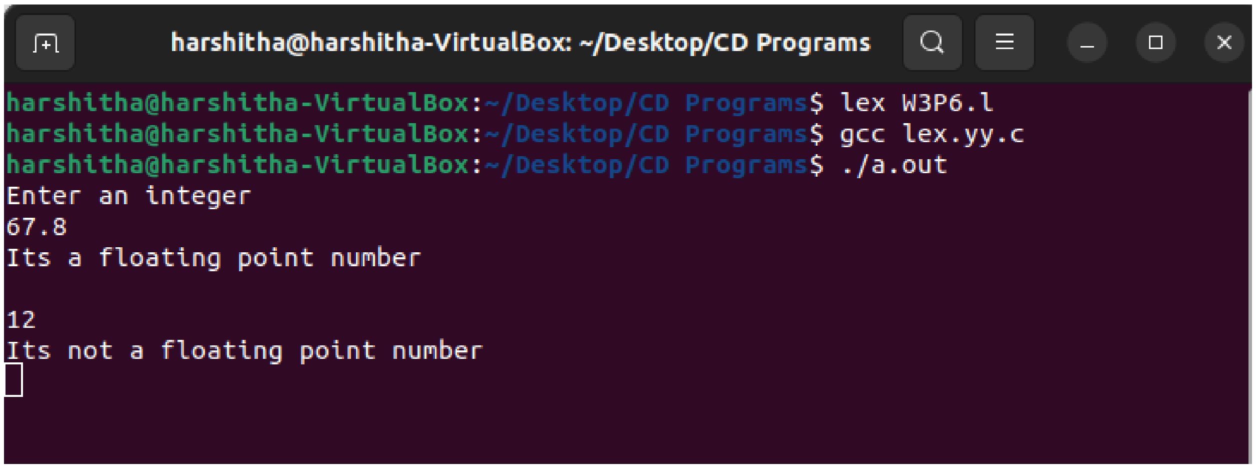
PROGRAM 17

Write a program in LEX to recognize Floating Point Numbers.

Code:

```
%option noyywrap
%{
  #include<stdio.h>
%}
%%
[+-]?[0-9]+[.][0-9]+ {printf("Its a floating point number \n");}
[+-]?[0-9]+ {printf("Its not a floating point number");}
%%
void main()
{
 printf("Enter an integer\n");
 yylex();
}
```

Output:



```
harshitha@harshitha-VirtualBox:~/Desktop/CD Programs$ lex W3P6.l
harshitha@harshitha-VirtualBox:~/Desktop/CD Programs$ gcc lex.yy.c
harshitha@harshitha-VirtualBox:~/Desktop/CD Programs$ ./a.out
Enter an integer
67.8
Its a floating point number

12
Its not a floating point number
```

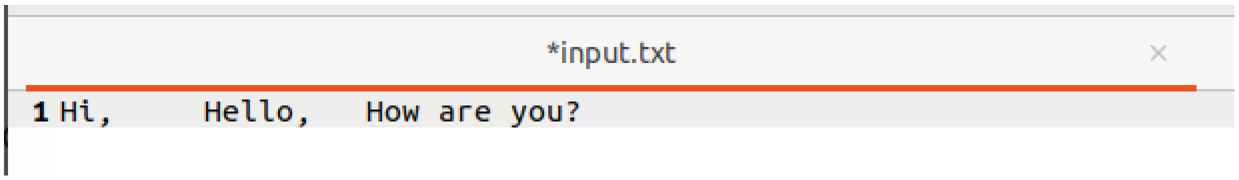
PROGRAM 18

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

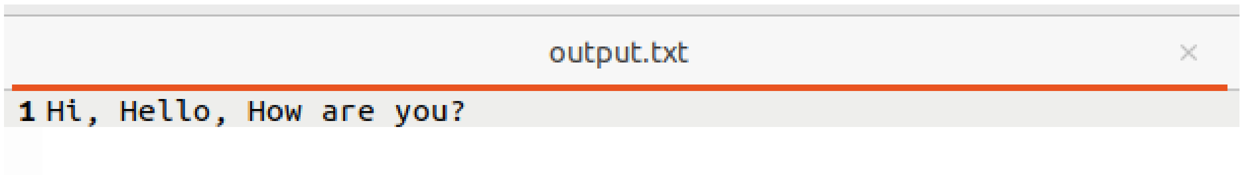
Code:

```
%option noyywrap
%{
#include<stdio.h>
%}
%%
[ \t]+ {fprintf(yyout," ");}
.|\n {fprintf(yyout,"%s",yytext);}
%%
void main()
{
yyin=fopen("input.txt","r");
yyout=fopen("output.txt","w");
yylex();
fclose(yyin);
fclose(yyout);
printf("Printed!\n");
}
```

Output:



```
Hi,      Hello,    How are you?
```



```
Hi, Hello, How are you?
```

PROGRAM 19

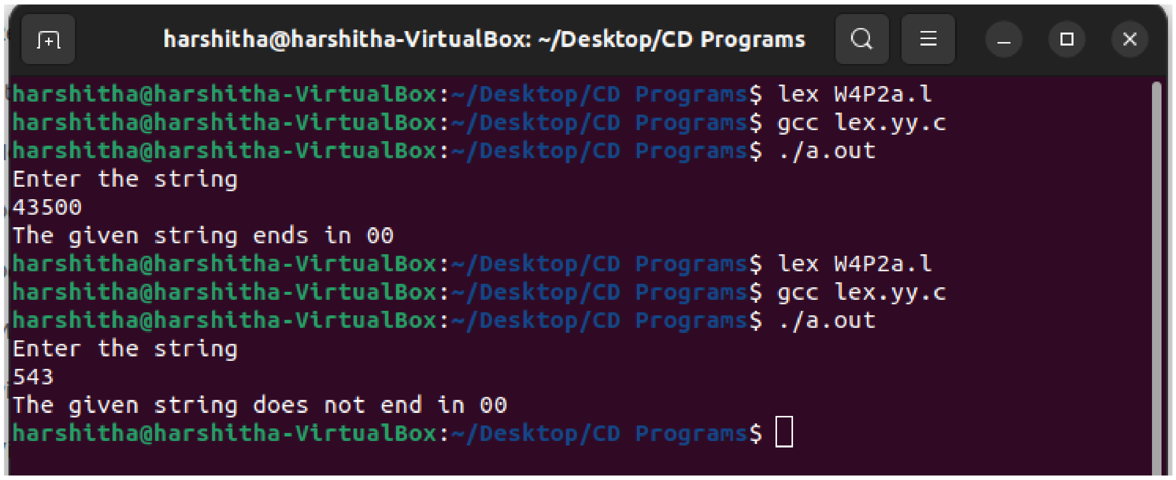
Write a LEX program to recognize the following tokens over the alphabets {0,1,..,9}

a) The set of all string ending in 00.

Code:

```
%option noyywrap
%{
  #include<stdio.h>
%}
%%
[0-9]*[0][0] {printf("The given string ends in 00 \n");}
.* {printf("The given string does not end in 00 \n");}
\n {return 0;}
%%
void main()
{
 printf("Enter the string \n");
 yylex();
}
```

Output:



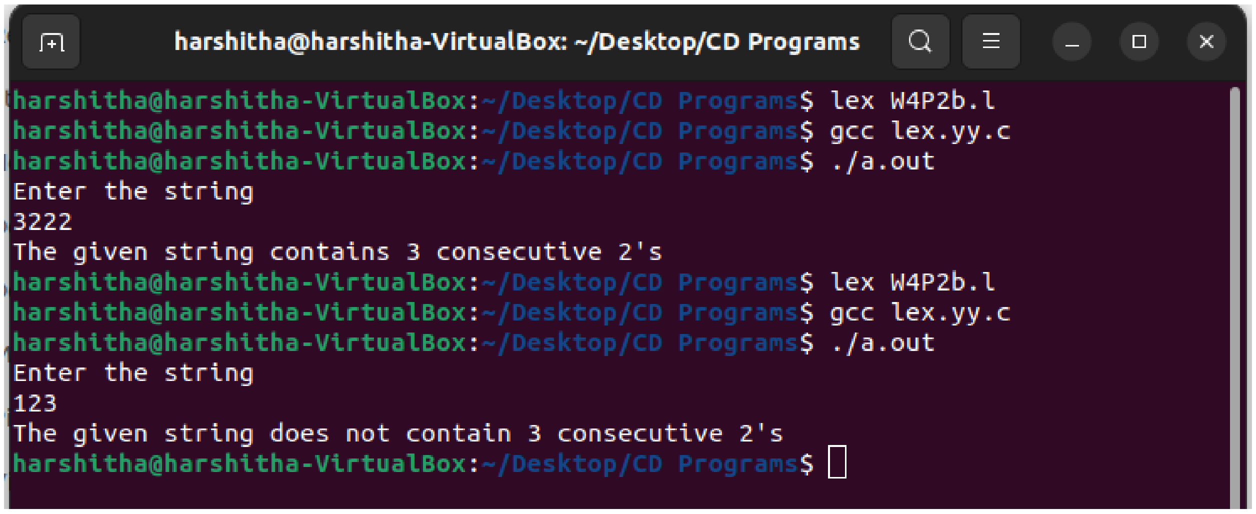
```
harshitha@harshitha-VirtualBox:~/Desktop/CD Programs$ lex W4P2a.l
harshitha@harshitha-VirtualBox:~/Desktop/CD Programs$ gcc lex.yy.c
harshitha@harshitha-VirtualBox:~/Desktop/CD Programs$ ./a.out
Enter the string
43500
The given string ends in 00
harshitha@harshitha-VirtualBox:~/Desktop/CD Programs$ lex W4P2a.l
harshitha@harshitha-VirtualBox:~/Desktop/CD Programs$ gcc lex.yy.c
harshitha@harshitha-VirtualBox:~/Desktop/CD Programs$ ./a.out
Enter the string
543
The given string does not end in 00
harshitha@harshitha-VirtualBox:~/Desktop/CD Programs$
```

b) The set of all strings with three consecutive 222’s.

Code:

```
%option noyywrap
%{
 #include<stdio.h>
%}
%%
[0-9]*[2][2][2][0-9]* {printf("The given string contains 3 consecutive 2's \n");}
.* {printf("The given string does not contain 3 consecutive 2's \n");}
\n {return 0;}
%%
void main()
{
 printf("Enter the string \n");
 yylex();
}
```

Output:



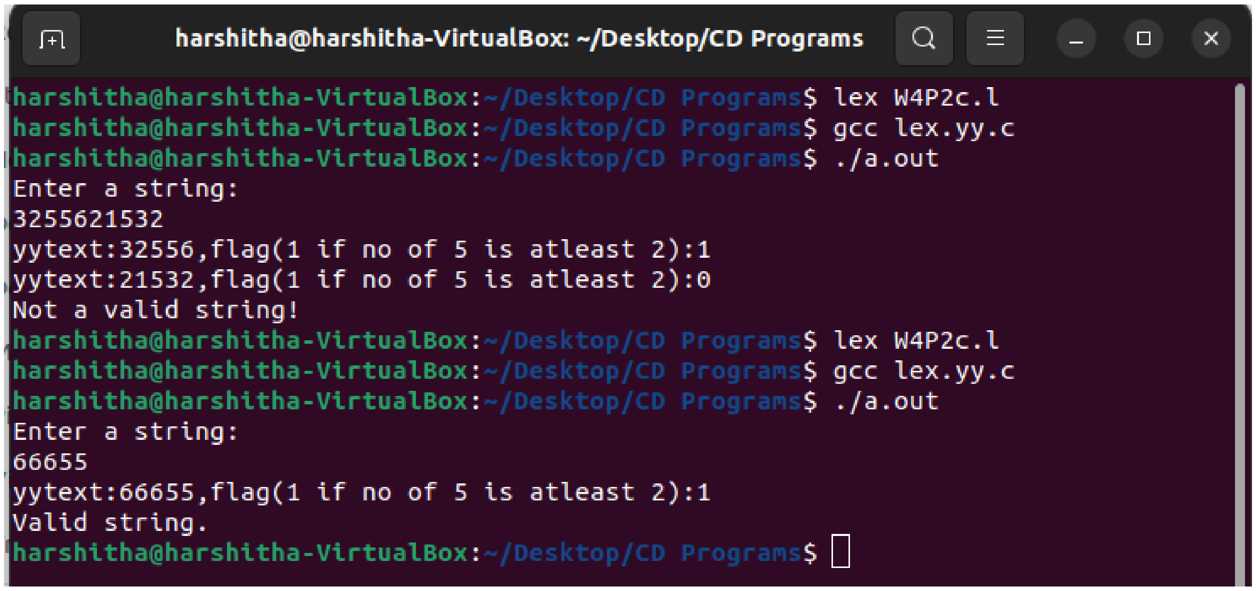
```
harshitha@harshitha-VirtualBox:~/Desktop/CD Programs$ lex W4P2b.l
harshitha@harshitha-VirtualBox:~/Desktop/CD Programs$ gcc lex.yy.c
harshitha@harshitha-VirtualBox:~/Desktop/CD Programs$ ./a.out
Enter the string
3222
The given string contains 3 consecutive 2's
harshitha@harshitha-VirtualBox:~/Desktop/CD Programs$ lex W4P2b.l
harshitha@harshitha-VirtualBox:~/Desktop/CD Programs$ gcc lex.yy.c
harshitha@harshitha-VirtualBox:~/Desktop/CD Programs$ ./a.out
Enter the string
123
The given string does not contain 3 consecutive 2's
harshitha@harshitha-VirtualBox:~/Desktop/CD Programs$
```

c) The set of all strings such that every block of five consecutive symbols contains at least two 5’s.

Code:

```
%{
#include<stdio.h>
int i,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;
        }
        }
        }
        count=0;
        printf("yytext:%s,flag(1 if no of 5 is atleast 2):%d\n",yytext,flag);
        if(flag!=1)
        {
        printf("Not a valid string!\n");
        return 0;
        }
        }

\n {return 0;}
%%
void main()
{
printf("Enter a string:\n");
yylex();
if(flag==1)
printf("Valid string.\n");
}
int yywrap()
{
return 1;
}
```

Output:



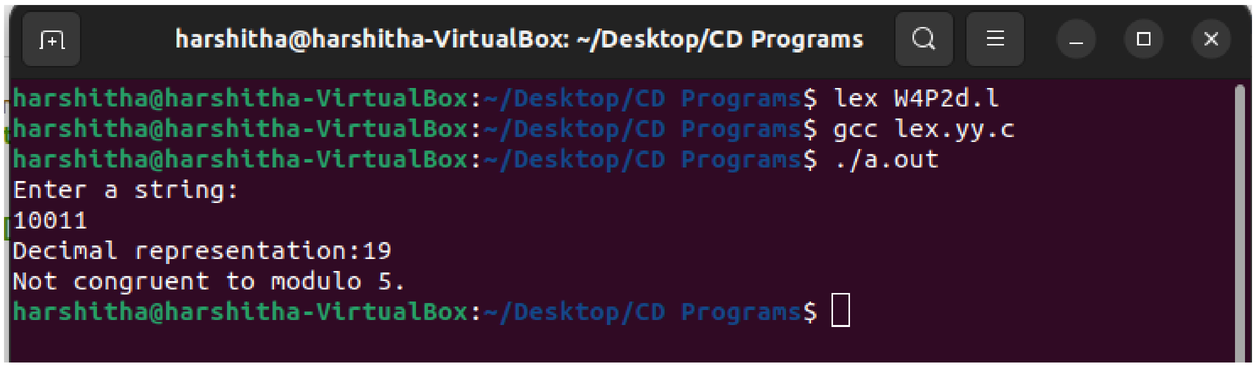
```
harshitha@harshitha-VirtualBox:~/Desktop/CD Programs$ lex W4P2c.l
harshitha@harshitha-VirtualBox:~/Desktop/CD Programs$ gcc lex.yy.c
harshitha@harshitha-VirtualBox:~/Desktop/CD Programs$ ./a.out
Enter a string:
3255621532
yytext:32556,flag(1 if no of 5 is atleast 2):1
yytext:21532,flag(1 if no of 5 is atleast 2):0
Not a valid string!
harshitha@harshitha-VirtualBox:~/Desktop/CD Programs$ lex W4P2c.l
harshitha@harshitha-VirtualBox:~/Desktop/CD Programs$ gcc lex.yy.c
harshitha@harshitha-VirtualBox:~/Desktop/CD Programs$ ./a.out
Enter a string:
66655
yytext:66655,flag(1 if no of 5 is atleast 2):1
Valid string.
harshitha@harshitha-VirtualBox:~/Desktop/CD Programs$ 
```

d) 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:

```
%{
#include<stdio.h>
int c,i,flag=1,sum=0,power=1;
%}
%%
^1[01]* {for(i=yyleng-1;i>=0;i--)
        {
        c=yytext[i]-'0';
        sum+=c*power;
        power*=2;
        }
        printf("Decimal representation:%d\n",sum);
        if(sum%5!=0)
        {
        printf("Not congruent to modulo 5.\n");
        sum=0;
        power=1;
        }
        else
        {
        printf("Congruent to modulo 5.\n");
        sum=0;
        power=1;
        }
        }
.* {printf("Not a binary number.\n");}
\n {return 0;}
%%
void main()
{
printf("Enter a string:\n");
yylex();
}
int yywrap()
{
return 1;}
```

Output:



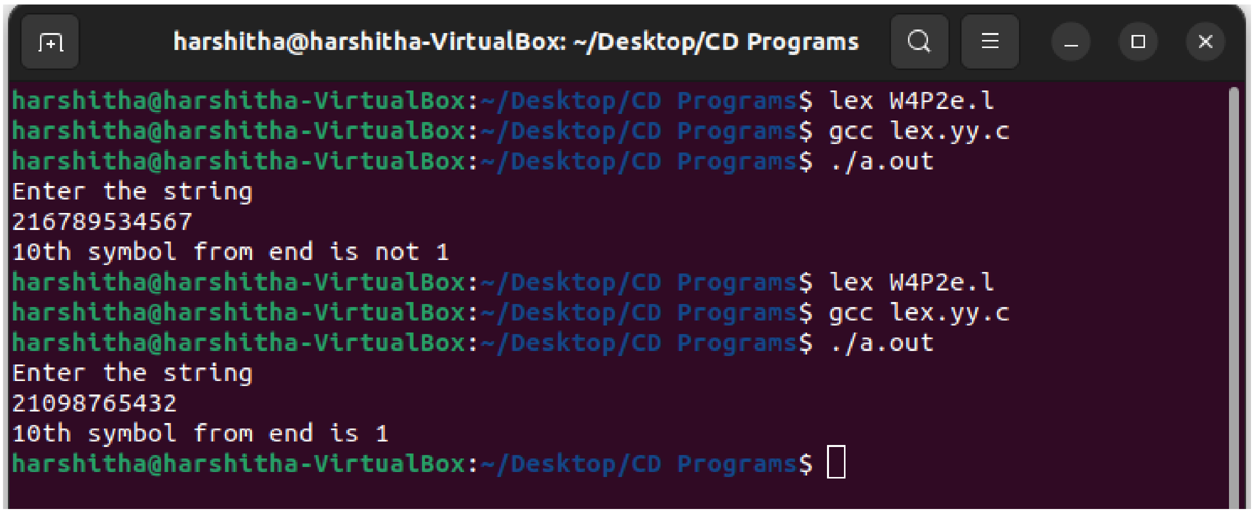
```
harshitha@harshitha-VirtualBox:~/Desktop/CD Programs$ lex W4P2d.l
harshitha@harshitha-VirtualBox:~/Desktop/CD Programs$ gcc lex.yy.c
harshitha@harshitha-VirtualBox:~/Desktop/CD Programs$ ./a.out
Enter a string:
10011
Decimal representation:19
Not congruent to modulo 5.
harshitha@harshitha-VirtualBox:~/Desktop/CD Programs$ 
```

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

Code:

```
%option noyywrap
%{
  #include<stdio.h>
%}
%%
[0-9]*[1][0-9][0-9][0-9][0-9][0-9][0-9][0-9][0-9][0-9] {printf("10th symbol from end is 1 \n");}
.* {printf("10th symbol from end is not 1\n");}
\n {return 0;}
%%
void main()
{
 printf("Enter the string \n");
 yylex();
}
```

Output:



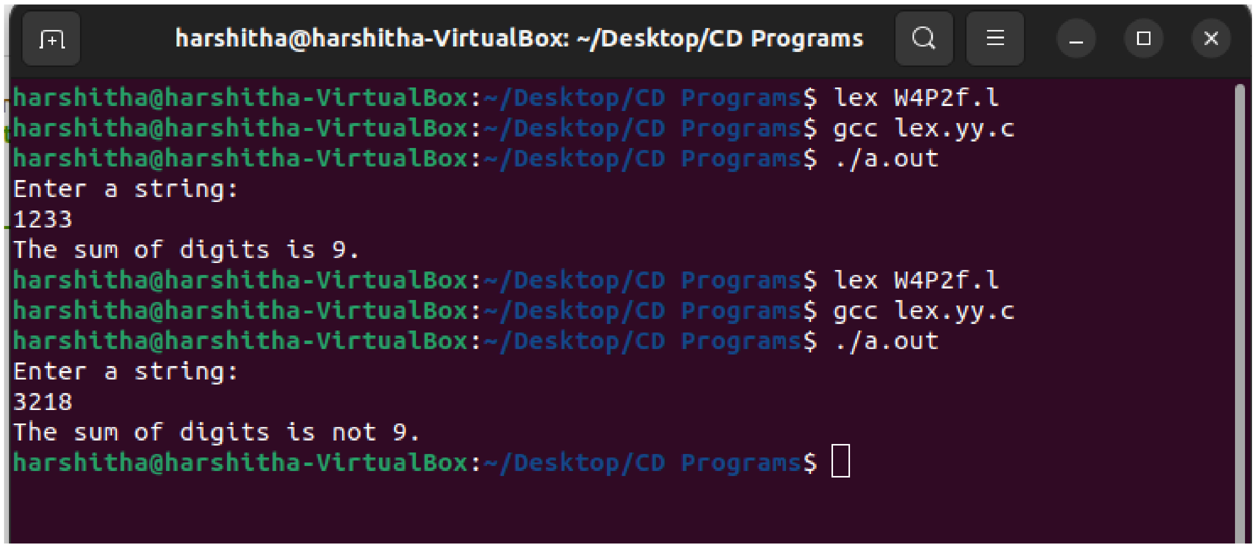
```
harshitha@harshitha-VirtualBox:~/Desktop/CD Programs$ lex W4P2e.l
harshitha@harshitha-VirtualBox:~/Desktop/CD Programs$ gcc lex.yy.c
harshitha@harshitha-VirtualBox:~/Desktop/CD Programs$ ./a.out
Enter the string
216789534567
10th symbol from end is not 1
harshitha@harshitha-VirtualBox:~/Desktop/CD Programs$ lex W4P2e.l
harshitha@harshitha-VirtualBox:~/Desktop/CD Programs$ gcc lex.yy.c
harshitha@harshitha-VirtualBox:~/Desktop/CD Programs$ ./a.out
Enter the string
21098765432
10th symbol from end is 1
harshitha@harshitha-VirtualBox:~/Desktop/CD Programs$
```

f) The set of all four digits numbers whose sum is 9

Code:

```
%{
#include<stdio.h>
int sum=0,i,flag=0;
%}
%%
[0-9][0-9][0-9][0-9] {for(i=0;i<yyleng;i++)
                    {
                    sum+=yytext[i]-'0';
                    }
                    if(sum==9)
                    {
                    flag=1;
                    sum=0;
                    }
                    else
                    {
                    flag=0;
                    sum=0;
                    }
                    }
\n {return 0;}
%%
void main()
{
printf("Enter a string:\n");
yylex();
if(flag==1)
printf("The sum of digits is 9.\n");
else
printf("The sum of digits is not 9.\n");
}
int yywrap()
{
return 1;
}
```

Output:



harshitha@harshitha-VirtualBox: ~/Desktop/CD Programs

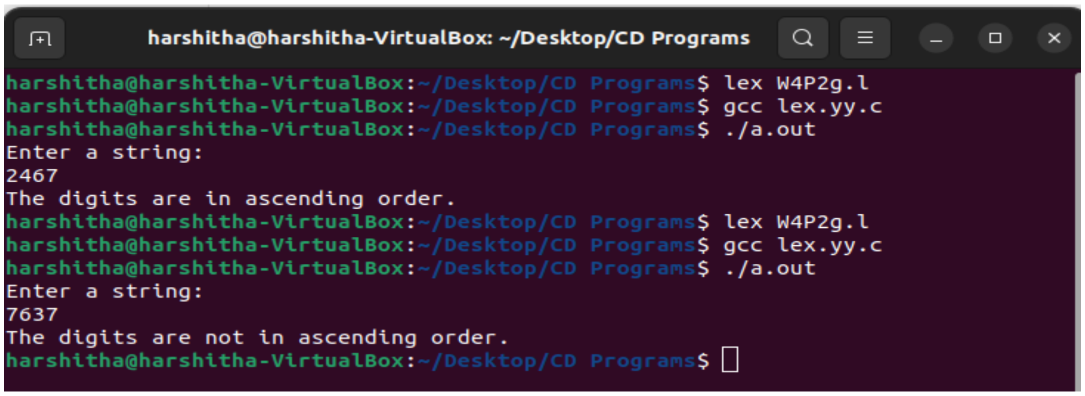
```
harshitha@harshitha-VirtualBox:~/Desktop/CD Programs$ lex W4P2f.l
harshitha@harshitha-VirtualBox:~/Desktop/CD Programs$ gcc lex.yy.c
harshitha@harshitha-VirtualBox:~/Desktop/CD Programs$ ./a.out
Enter a string:
1233
The sum of digits is 9.
harshitha@harshitha-VirtualBox:~/Desktop/CD Programs$ lex W4P2f.l
harshitha@harshitha-VirtualBox:~/Desktop/CD Programs$ gcc lex.yy.c
harshitha@harshitha-VirtualBox:~/Desktop/CD Programs$ ./a.out
Enter a string:
3218
The sum of digits is not 9.
harshitha@harshitha-VirtualBox:~/Desktop/CD Programs$ 
```

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

Code:

```
%option noyywrap
%{
#include<stdio.h>
int c,i,flag=1;
%}
%%
[0-9][0-9][0-9][0-9] {for(i=0;i<yyleng-1;i++)
                     {
                     if(yytext[i]>=yytext[i+1])
                     {
                     flag=0;
                     break;
                     }}}
\n {return 0;}
%%
void main()
{
printf("Enter a string:\n");
yylex();
if(flag==1)
printf("The digits are in ascending order.\n");
else
printf("The digits are not in ascending order.\n");
}
```

Output:



```
harshitha@harshitha-VirtualBox:~/Desktop/CD Programs$ lex W4P2g.l
harshitha@harshitha-VirtualBox:~/Desktop/CD Programs$ gcc lex.yy.c
harshitha@harshitha-VirtualBox:~/Desktop/CD Programs$ ./a.out
Enter a string:
2467
The digits are in ascending order.
harshitha@harshitha-VirtualBox:~/Desktop/CD Programs$ lex W4P2g.l
harshitha@harshitha-VirtualBox:~/Desktop/CD Programs$ gcc lex.yy.c
harshitha@harshitha-VirtualBox:~/Desktop/CD Programs$ ./a.out
Enter a string:
7637
The digits are not in ascending order.
harshitha@harshitha-VirtualBox:~/Desktop/CD Programs$
```

PROGRAM 20

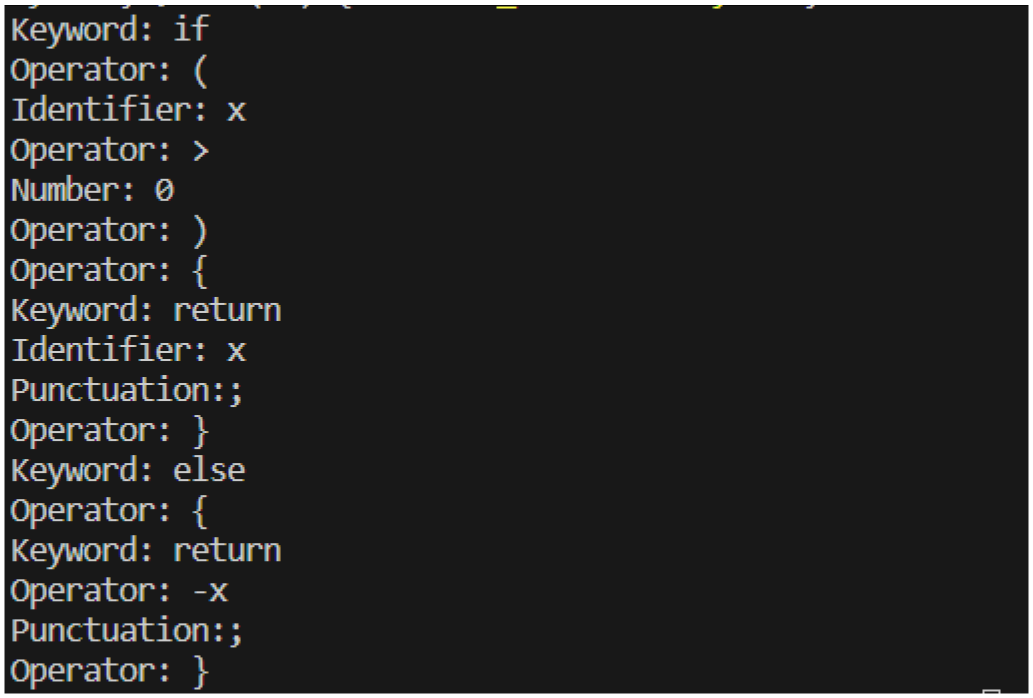
Write a program to design Lexical Analyzer to recognize any five keywords, identifiers, numbers, operators and punctuations.

Code:

```
#include <stdio.h>

bool isPunctuator(char ch)
{
   if (ch == ' ' || ch == '+' || ch == '-' || ch == '*' ||
       ch == '/' || ch == ',' || ch == ';' || ch == '>' ||
       ch == '<' || ch == '=' || ch == '(' || ch == ')' ||
       ch == '[' || ch == ']' || ch == '{' || ch == '}')
       return (true);
   return (false);
}


bool isOperator(char ch)
{
   if (ch == '+' || ch == '-' || ch == '*' ||
       ch == '/' || ch == '>' || ch == '<' ||
       ch == '=')
       return (true);
   return (false);
}

bool validIdentifier(char* str)
{
   if (str[0] == '0' || str[0] == '1' || str[0] == '2' ||
       str[0] == '3' || str[0] == '4' || str[0] == '5' ||
       str[0] == '6' || str[0] == '7' || str[0] == '8' ||
       str[0] == '9' || isPunctuator(str[0]) == true)
       return (false);
   return (true);
}

bool isKeyword(char* str)
```

```
{
    if (!strcmp(str, "if") || !strcmp(str, "else") ||
        !strcmp(str, "while") || !strcmp(str, "do") ||
        !strcmp(str, "break") ||
         !strcmp(str, "continue") || !strcmp(str, "int")
        || !strcmp(str, "double") || !strcmp(str, "float")
        || !strcmp(str, "return") || !strcmp(str, "char")
        || !strcmp(str, "case") || !strcmp(str, "char")
        || !strcmp(str, "sizeof") || !strcmp(str, "long")
        || !strcmp(str, "short") || !strcmp(str, "typedef")
        || !strcmp(str, "switch") || !strcmp(str, "unsigned")
        || !strcmp(str, "void") || !strcmp(str, "static")
        || !strcmp(str, "struct") || !strcmp(str, "goto"))
        return (true);
    return (false);
}

bool isNumber(char ch)
{
    if (ch == '0' || ch == '1' || ch == '2' || ch == '3' ||
        ch == '4' || ch == '5' || ch == '6' || ch == '7' ||
        ch == '8' || ch == '9')
        return (true);
    return (false);
}


void parse(char* str) {
    int left = 0, right = 0;
    int len = strlen(str);

    while (right <= len) {
        if (!isDelimiter(str[right]))
            right++;

        if (isDelimiter(str[right]) || right == len) {
            char* subStr = subString(str, left, right - 1);

            if (right != len && isOperator(str[right]))
                printf("'%c' IS AN OPERATOR\n", str[right]);
            else if (isKeyword(subStr))
                printf("'%s' IS A KEYWORD\n", subStr);
            else if (isInteger(subStr))
                printf("'%s' IS AN INTEGER\n", subStr);
            else if (isRealNumber(subStr))
                printf("'%s' IS A REAL NUMBER\n", subStr);
            else if (validIdentifier(subStr))
                printf("'%s' IS A VALID IDENTIFIER\n", subStr);
            else
                printf("'%s' IS NOT A VALID IDENTIFIER\n", subStr);

            left = ++right;
        }
    }
}

int main() {
    char str[100] = "int a = b + 1c; ";
    parse(str);
    return 0;
}
```

Output:



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

PROGRAM 21

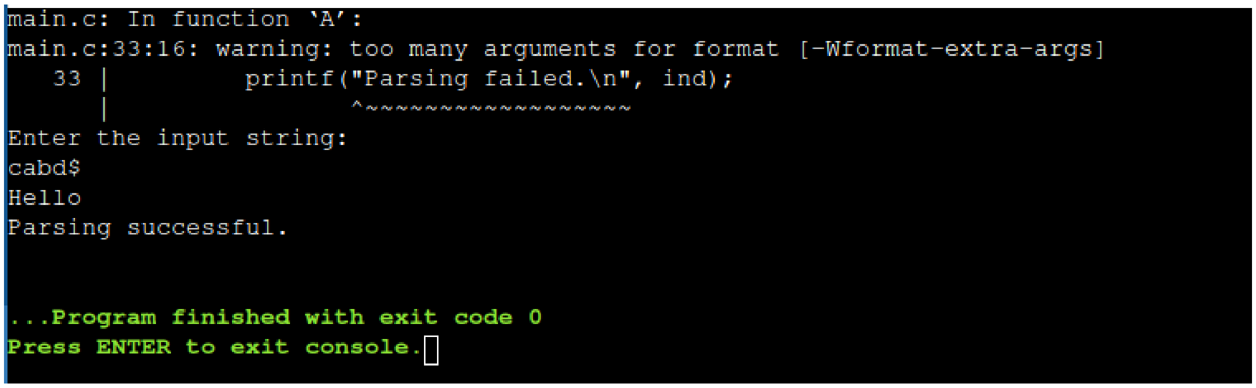
Write a Program to perform Recursive Descent Parsing on the following grammar.
S->cAd , A->ab /a

Code:

```
#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 if (input[ind] == 'a')
	{
	printf("Hi!\n");
	match('a');
	}*/
	else
	{
	printf("Parsing failed.\n", ind);
	exit(1);
```

```
    }
}
int main() {
    printf("Enter the 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:



```
main.c: In function ‘A’:
main.c:33:16: warning: too many arguments for format [-Wformat-extra-args]
   33 |         printf("Parsing failed.\n", ind);
      |                ^~~~~~~~~~~~~~~~~~~
Enter the input string:
cabd$
Hello
Parsing successful.


...Program finished with exit code 0
Press ENTER to exit console.
```

PROGRAM 22

Write a YACC program to implement desk calculator

Code:

calci.l

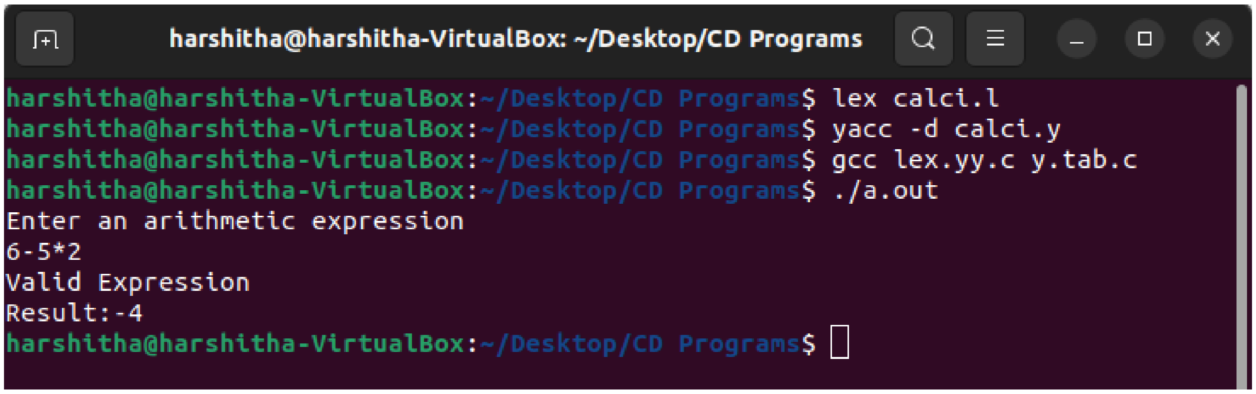
```
%{
  #include<stdio.h>
  #include<stdlib.h>
  #include "y.tab.h"
  extern int yylval;
%}
%%
[0-9]+ {yylval=atoi(yytext);return num;}
[\t] ;
\n {return 0;}
. {return yytext[0];}
%%
int yywrap()
{
}
```

calci.y

```
%{
  #include<stdio.h>
  #include<stdlib.h>
  int yyerror(const char *s);
  int yylex(void);
%}
%token num;
%left '+' '-'
%left '*' '/'
%left ')'
%left '('
%%
s:e {printf("Valid Expression \n");
            printf("Result:%d\n",$$);
            exit(0);
            };
```

```
e:e'+'e {$$=$1+$3;}
 |e'-'e {$$=$1-$3;}
 |e'*'e {$$=$1*$3;}
 |e'/'e {$$=$1/$3;}
 |'('e')' {$$=$2;}
 |num {$$=$1;}
 ;
%%

void main()
{
 printf("Enter an arithmetic expression \n");
 yyparse();
}
int yyerror(const char *s)
{
 printf("Invalid expression \n");
 return 0;
}
```

Output:



```
harshitha@harshitha-VirtualBox:~/Desktop/CD Programs$ lex calci.l
harshitha@harshitha-VirtualBox:~/Desktop/CD Programs$ yacc -d calci.y
harshitha@harshitha-VirtualBox:~/Desktop/CD Programs$ gcc lex.yy.c y.tab.c
harshitha@harshitha-VirtualBox:~/Desktop/CD Programs$ ./a.out
Enter an arithmetic expression
6-5*2
Valid Expression
Result:-4
harshitha@harshitha-VirtualBox:~/Desktop/CD Programs$ 
```

PROGRAM 23

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

Code:

syntaxtree.l

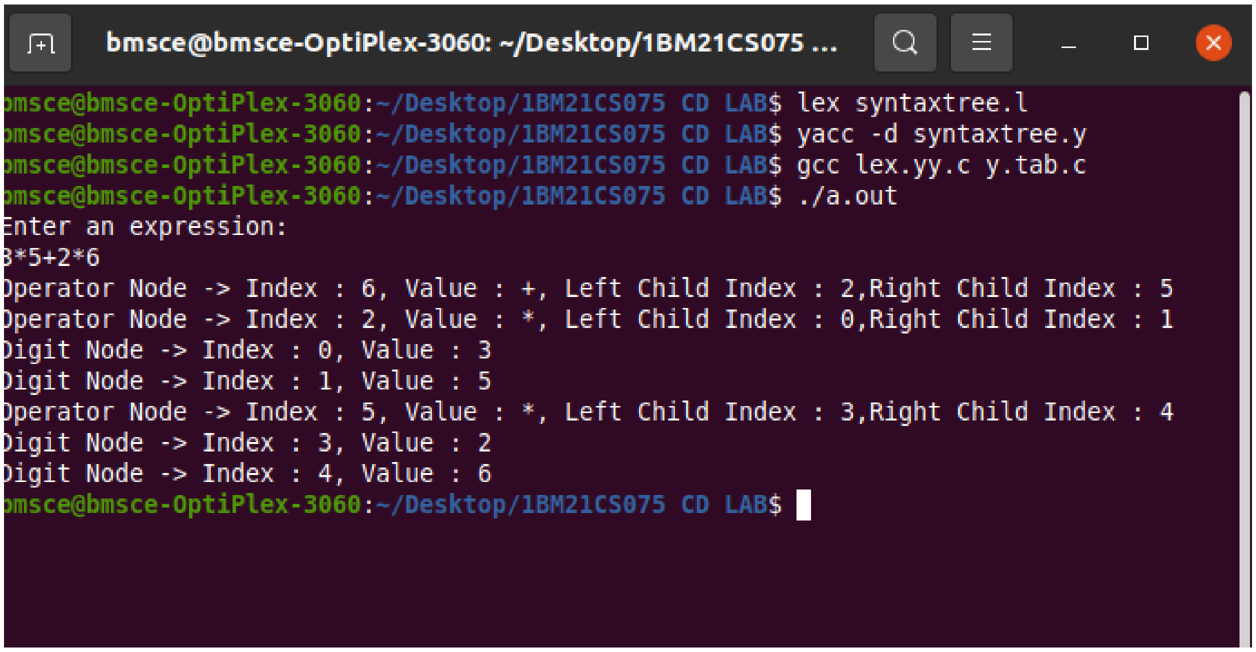
```
%{
#include<stdio.h>
#include<stdlib.h>
#include "y.tab.h"
extern int yylval;
%}
%%
[0-9]+ {yylval=atoi(yytext);return digit;}
[\t] ;
[\n] return 0;
. return yytext[0];
%%
int yywrap()
{
return 1;
}
```

syntaxtree.y

```
%{
#include <math.h>
#include<ctype.h>
#include<stdio.h>
#include<stdlib.h>
#include<string.h>
int yyerror(char *s);
int yylex(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 {$$=mknode($1,$3,"+");}
|T {$$=$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;
}
```

```
/*my_print_tree function to print the syntax tree in DLR fashion*/
void my_print_tree(int cur_ind)
{
if(cur_ind==-1) return;
if(syn_tree[cur_ind].lc==-1&&syn_tree[cur_ind].rc==-1)
printf("Digit Node -> Index : %d, Value : %s\n",cur_ind,syn_tree[cur_ind].val);
else
printf("Operator Node -> Index : %d, Value : %s, Left Child Index : %d,Right Child 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(syn_tree[cur_ind].rc);
}
```

Output:



```
omsce@bmsce-OptiPlex-3060:~/Desktop/1BM21CS075 CD LAB$ lex syntaxtree.l
omsce@bmsce-OptiPlex-3060:~/Desktop/1BM21CS075 CD LAB$ yacc -d syntaxtree.y
omsce@bmsce-OptiPlex-3060:~/Desktop/1BM21CS075 CD LAB$ gcc lex.yy.c y.tab.c
omsce@bmsce-OptiPlex-3060:~/Desktop/1BM21CS075 CD LAB$ ./a.out
Enter an expression:
3*5+2*6
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 : 3
Digit Node -> Index : 1, Value : 5
Operator Node -> Index : 5, Value : *, Left Child Index : 3,Right Child Index : 4
Digit Node -> Index : 3, Value : 2
Digit Node -> Index : 4, Value : 6
omsce@bmsce-OptiPlex-3060:~/Desktop/1BM21CS075 CD LAB$
```

PROGRAM 24

Use YACC to convert: Infix expression to Postfix expression.

Code:

infixtopostfix.l

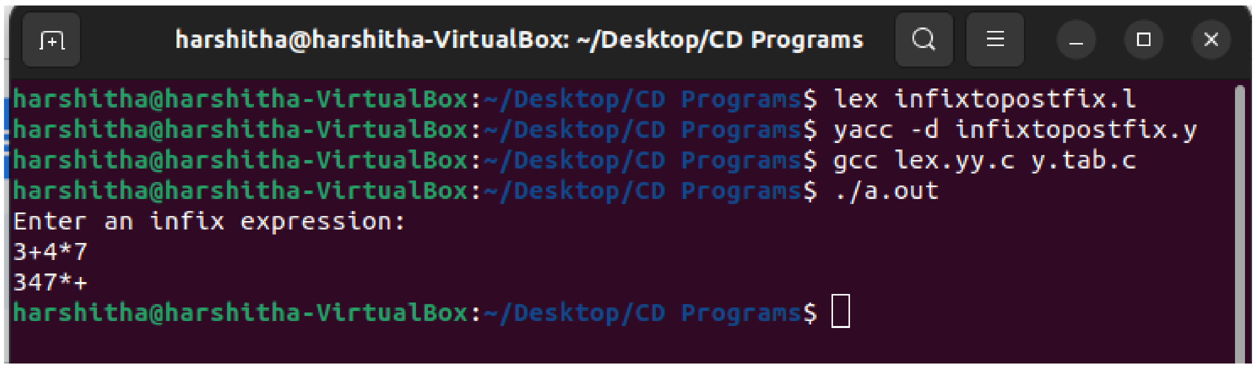
```
%{
#include<stdio.h>
#include<stdlib.h>
#include "y.tab.h"
extern int yylval;
%}
%%
[0-9]+ {yylval=atoi(yytext);return num;}
[\t ] ;
\n {return 0;}
. {return yytext[0];}
%%
int yywrap()
{
}
```

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 an infix expression:\n");
yyparse();
}
int yyerror(const char *s)
{
printf("Invalid infix expression!\n");
return 0;
}
```

Output:



```
harshitha@harshitha-VirtualBox:~/Desktop/CD Programs$ lex infixtopostfix.l
harshitha@harshitha-VirtualBox:~/Desktop/CD Programs$ yacc -d infixtopostfix.y
harshitha@harshitha-VirtualBox:~/Desktop/CD Programs$ gcc lex.yy.c y.tab.c
harshitha@harshitha-VirtualBox:~/Desktop/CD Programs$ ./a.out
Enter an infix expression:
3+4*7
347*+
harshitha@harshitha-VirtualBox:~/Desktop/CD Programs$
```

PROGRAM 25

Write a YACC program to recognize the grammar ($a^n b$, n>=5)

Code:

stringmatch.l

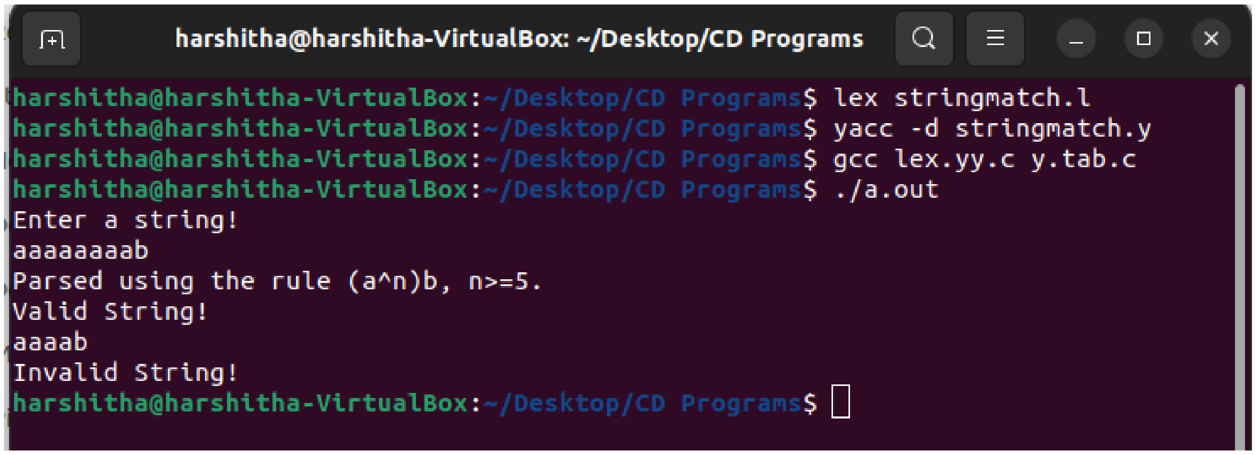
```
%{
#include<stdio.h>
#include<stdlib.h>
#include "y.tab.h"
extern int yylval;
%}
%%
[aA] {yylval=yytext[0];return A;}
[bB] {yylval=yytext[0];return B;}
\n {return NL;}
. {return yytext[0];}
%%
int yywrap()
{
return 1;
}
```

stringmatch.y

```
%{
#include<stdio.h>
#include<stdlib.h>
int yyerror(char *s);
int yylex(void);
%}
%token A
%token B
%token NL
%%
smtr:A A A A A S B NL {printf("Parsed using the rule (a^n)b, n>=5.\nValid String!\n");}
;
S:S A
```

```
|
;
%%
void main()
{
printf("Enter a string!\n");
yyparse();
}
int yyerror(char *s)
{
printf("Invalid String!\n");
return 0;
}
```

Output:



```
harshitha@harshitha-VirtualBox:~/Desktop/CD Programs$ lex stringmatch.l
harshitha@harshitha-VirtualBox:~/Desktop/CD Programs$ yacc -d stringmatch.y
harshitha@harshitha-VirtualBox:~/Desktop/CD Programs$ gcc lex.yy.c y.tab.c
harshitha@harshitha-VirtualBox:~/Desktop/CD Programs$ ./a.out
Enter a string!
aaaaaaaab
Parsed using the rule (a^n)b, n>=5.
Valid String!
aaaab
Invalid String!
harshitha@harshitha-VirtualBox:~/Desktop/CD Programs$
```

PROGRAM 26

Use YACC to generate 3-Address code for a given expression

Code:

addresscode.l

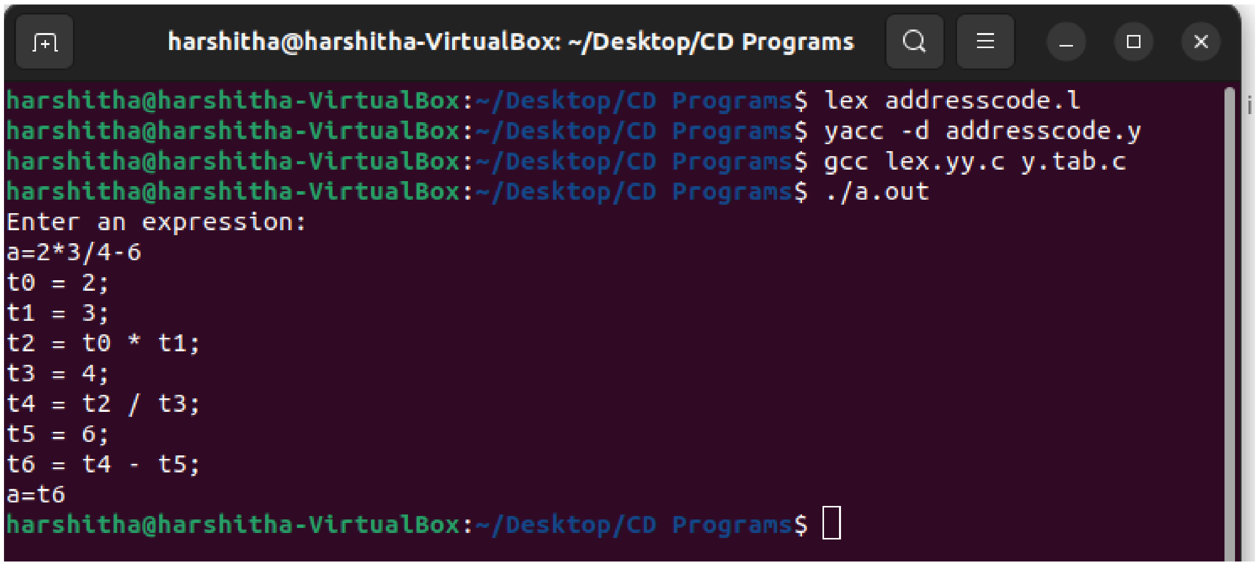
```
%{
#include<stdio.h>
#include<stdlib.h>
#include"y.tab.h"
extern int yylval;
extern char iden[20];
%}
d [0-9]+
a [a-zA-Z]+
%%
{d} { yylval=atoi(yytext); return digit; }
{a} { strcpy(iden,yytext); yylval=1; return id;}
[ \t] {;}
\n return 0;
. return yytext[0];
%%
int yywrap()
{
return 1;
}
```

addresscode.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("%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;}
;
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);}
;
%%
int main()
{
var_cnt=0;
printf("Enter an expression:\n");
yyparse();
return 0;
}
int yyerror(char *s)
{
printf("Invalid expression!");
return 0;
}
```

Output:



harshitha@harshitha-VirtualBox: ~/Desktop/CD Programs

```
harshitha@harshitha-VirtualBox:~/Desktop/CD Programs$ lex addresscode.l
harshitha@harshitha-VirtualBox:~/Desktop/CD Programs$ yacc -d addresscode.y
harshitha@harshitha-VirtualBox:~/Desktop/CD Programs$ gcc lex.yy.c y.tab.c
harshitha@harshitha-VirtualBox:~/Desktop/CD Programs$ ./a.out
Enter an expression:
a=2*3/4-6
t0 = 2;
t1 = 3;
t2 = t0 * t1;
t3 = 4;
t4 = t2 / t3;
t5 = 6;
t6 = t4 - t5;
a=t6
harshitha@harshitha-VirtualBox:~/Desktop/CD Programs$ 
```