SAVEETHA SCHOOL OF ENGINEERING

CSA1404

COMPILER DESIGN LAB MANUAL

Shivaji V

192424370

Q1) Develop a lexical Analyzer to identify identifiers, constants, operators using C program.

Output

```
enter the string : a=b+c*d-9
 identifiers : a b c d
 constants : 9
 operators : = + * -
```

Q2) Develop a lexical Analyzer to identify whether a given line is a comment or not using C.

Output

```
Enter comment:// Hello World

It is a comment
```

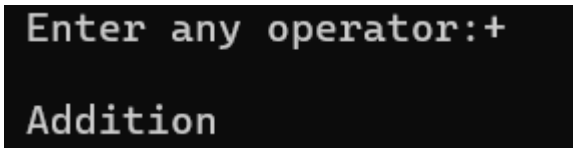
Q3) Design a lexical Analyzer for given language should ignore the redundant spaces, tabs and new lines and ignore comments using C.

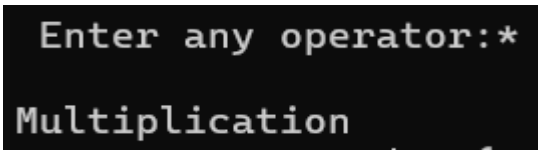
Output

```
define is identifier
PI is identifier
314 is identifier
includestdioh is identifier
includeconioh is identifier
void is keyword
main is keyword
int is keyword
abc is identifier
= is operator
30 is identifier
printfhello is identifier
```

Q4) Design a lexical Analyzer to validate operators to recognize the operators +,-,*,/ using regular arithmetic operators using C.

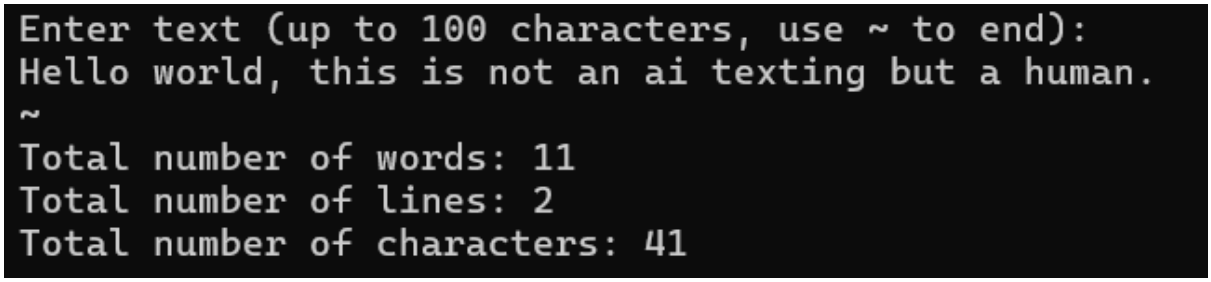
Output





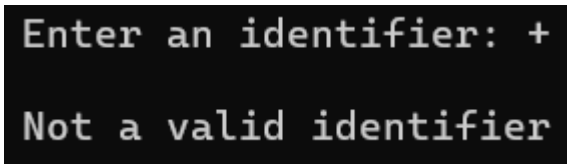
Q5) Design a lexical Analyzer to find the number of whitespaces and newline characters using C.

Output



Q6) Develop a lexical Analyzer to test whether a given identifier is valid or not using C.

Output



Q7) Write a C program to find FIRST( ) - predictive parser for the given grammar

Output

```
How many number of productions ? :4
Enter productions Number 1 : S=AaBb
Enter productions Number 2 : S=Ba
Enter productions Number 3 : A=$
Enter productions Number 4 : B=$

 Find the FIRST of :S

 FIRST(S)= {  $  a }
press 'y' to continue : y

 Find the FIRST of :A

 FIRST(A)= {  $ }
```

Q8) Write a C program to find FOLLOW( ) - predictive parser for the given grammar.

Output

```
Enter Total Number of Productions:      3

Value of Production Number [1]: S=Aab

Value of Production Number [2]: A=+Ab

Value of Production Number [3]: B=bB

Enter production Value to Find Follow:  S

Follow Value of S:      { $ }
To Continue, Press Y:   Y

Enter production Value to Find Follow:  A

Follow Value of A:      { a b }
```

Q9) Implement a C program to eliminate left recursion from a given CFG.

Output

```
Enter Number of Production : 3
Enter the grammar as E->E-A :
S->(L)|a
L->L,S|S
L->b

GRAMMAR : : : S->(L)|a is not left recursive.

GRAMMAR : : : L->L,S|S is left recursive.
Grammar without left recursion:
L->SL'
L'->,L'|E

GRAMMAR : : : L->b is not left recursive.

Process returned 0 (0x0)   execution time : 57.803 s
Press any key to continue.
```

Q10) Implement a C program to eliminate left factoring from a given CFG.

Output

```
Enter Production: S->iEtS|iEtSeS|a

 S->iEtSX
 X->|eS|a
```

Q11) Implement a C program to perform symbol table operations.

Output

```
1.insert
2.display
3.search
4.modify
5.exit
1
enter the label a
enter the address 100

1.insert
2.display
3.search
4.modify
5.exit
2
a       100

1.insert
2.display
3.search
4.modify
5.exit
3
enter the label a
label is found
1.insert
2.display
3.search
4.modify
5.exit
4
enter the labe: a
label is found
enter the address 200
```

Q12) Write a C program to construct recursive descent parsing for the given grammar.

Output

```
Recursive descent parsing for the grammar:
E -> T E'
E' -> + T E' | @
T -> F T'
T' -> * F T' | @
F -> (E) | ID

Enter the string to be checked: (a+b)*c

String is accepted
```

Q13) Write a C program to implement either Top Down parsing technique or Bottom Up Parsing technique to check whether the given input string is satisfying the grammar or not.

Output

```
The grammar is: S->aS, S->Sb, S->ab
Enter the string to be checked:
abb
String accepted
```

Q14) Implement the concept of Shift reduce parsing in C Programming.

Output

```
                  SHIFT REDUCE PARSER

 GRAMMER

 E->E+E
 E->E/E
 E->E*E
 E->a/b
 enter the input symbol:         a+b

          stack implementation table
 stack             input symbol              action


 $                a+b$                       --
 $a                +b$                       shift a
 $E                +b$                       E->a
 $E+                b$                       shift+
 $E+b               $                        shiftb
 $E+E               $                        E->b
 $E                 $                        E->E+E
 $E                 $                        ACCEPT
```

Q15) Write a C Program to implement the operator precedence parsing.

Output

```
Enter the string
i*(i+i)*i

STACK    INPUT    ACTION
$i       *(i+i)*i$        Shift
$E       *(i+i)*i$        Reduced: E->i
$E*      (i+i)*i$         Shift
$E*(     i+i)*i$ Shift
$E*(i    +i)*i$  Shift
$E*(E    +i)*i$  Reduced: E->i
$E*(E+   i)*i$   Shift
$E*(E+i  )*i$    Shift
$E*(E+E  )*i$    Reduced: E->i
$E*(E    )*i$    Reduced: E->E+E
$E*(E)   *i$     Shift
$E*E     *i$     Reduced: E->)E(
$E       *i$     Reduced: E->E*E
$E*      i$      Shift
$E*i     $       Shift
$E*E     $       Reduced: E->i
$E       $       Reduced: E->E*E
$E$              Shift
$E$              Shift
Accepted;
```

Q16) Write a C Program to Generate the Three address code representation for the given input statement.

Output

```
THREE ADDRESS CODE:
t1 = b + c
t2 = t1 - d
t3 = t2 * e

a = t3
```

Q17) Write a C program for implementing a Lexical Analyzer to Scan and Count the number of characters, words, and lines in a file.

Output

```
void main()
{
int a;
int b;
a = b + c;
c = d * e;
}~
Total number of words : 18
Total number of lines : 7
Total number of characters : 34
```

Q18) Write a C program to implement the back end of the compiler.

Output

```
enter the no: intermediate code:2
enter the 3 address code:1:a=b+c
enter the 3 address code:2:d=n*b
the generated code is:
 mov b,R0
 add c,R0
 mov R0,a

 mov n,R1
 mul b,R1
 mov R1,d
```

Q19) Write a C program to compute LEADING( ) – operator precedence parser for the given grammar.

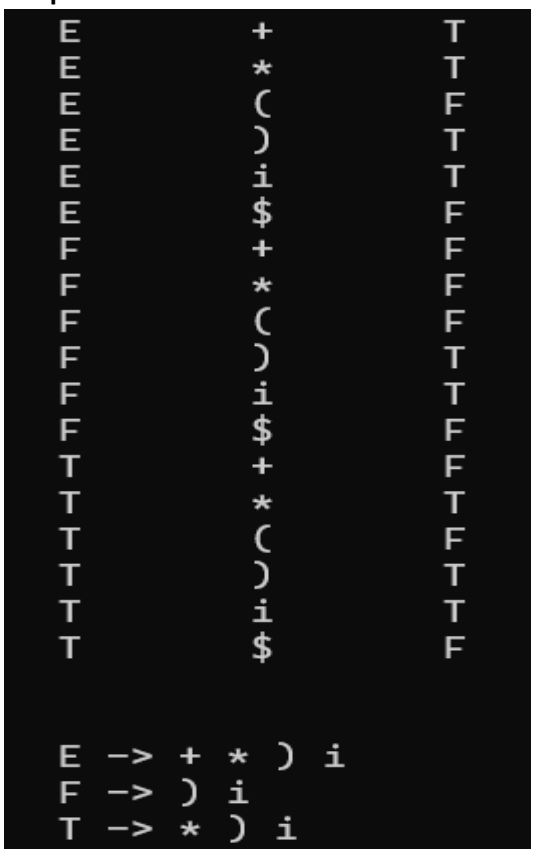
Output

```
E        +        T
E        *        T
E        (        T
E        )        F
E        i        T
E        $        F
F        +        F
F        *        F
F        (        T
F        )        F
F        i        T
F        $        F
T        +        F
T        *        T
T        (        T
T        )        F
T        i        T
T        $        F


E -> + * ( i
F -> ( i
T -> * ( i
```

Q20) Write a C program to compute TRAILING( ) – operator precedence parser for the given grammar.

Output



Q21) Write a LEX specification file to take input C program from a .c file and count the number of characters, number of lines & number of words.

```
%{
int nchar, nword, nline;
%}
%%
\n { nline++; nchar++; }
[^ \t\n]+ { nword++, nchar += yyleng; }
. { nchar++; }
%%
int yywrap(void) {
return 1;
}
int main(int argc, char *argv[]) {
yyin = fopen(argv[1], "r");
yylex();
printf("Number of characters = %d\n", nchar);
printf("Number of words = %d\n", nword);
printf("Number of lines = %d\n", nline);
fclose(yyin);
}
```

Output

```
C:\Users\Shivaji V\LexPrograms>count.exe input.txt
Number of characters = 40
Number of words = 8
Number of lines = 3
```

Q22) Write a LEX program to print all the constants in the given C source program file.

```
%{
int cons = 0;
%}
%%
[0-9]+     { cons++; printf("%s is a constant\n", yytext); }
.|\n       { }
%%
int yywrap(void) { return 1; }
int main(void)
{
   FILE *f;
   char file[50];
   printf("Enter File Name : ");
   scanf("%s", file);
   f = fopen(file, "r");
   yyin = f;
   yylex();
   printf("Number of Constants : %d\n", cons);
   fclose(f);
   return 0;
}
```

Output

```
Enter File Name : number.txt
10 is a constant
20 is a constant
12345 is a constant
Number of Constants : 3
```

Q23) Write a LEX program to count the number of Macros defined and header files included in the C program.

```
%{
#include <stdio.h>
int macros = 0, headers = 0;
%}
%%
"#define"    { macros++; }
"#include"   { headers++; }
.|\n        { }
%%
int yywrap() { return 1; }
int main(int argc, char *argv[]) {
   yyin = fopen(argv[1], "r");
   yylex();
   printf("Macros  = %d\n", macros);
   printf("Headers = %d\n", headers);
   return 0;
}
```

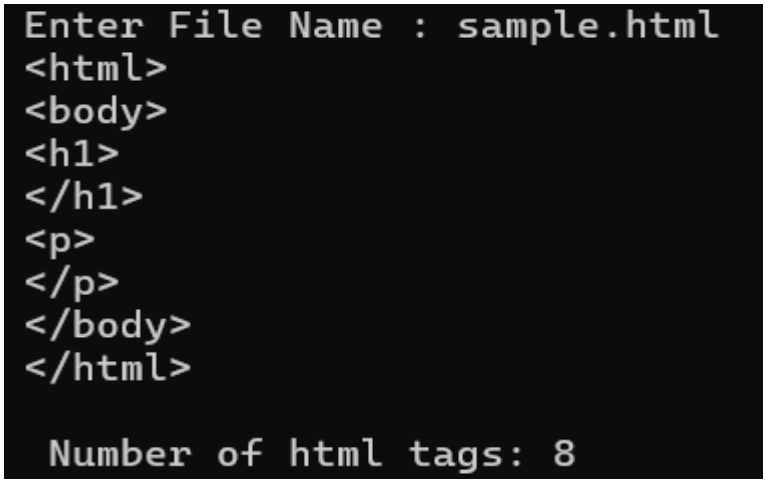
Output

```
C:\Users\Shivaji V\LexPrograms>macros.exe macros.txt
Number of macros defined = 1
Number of header files included = 2
```

Q24) Write a LEX program to print all HTML tags in the input file

```
%{
int tags;
%}
%%
"<"[^>]*> { tags++; printf("%s \n", yytext); }
.|\n { }
%%
int yywrap(void) {
return 1; }
int main(void)
{
FILE *f;
char file[10];
printf("Enter File Name : ");
scanf("%s",file);
f = fopen(file,"r");
yyin = f;
yylex();
printf("\n Number of html tags: %d",tags);
fclose(yyin);
}
```

Output



Q25) Write a LEX program which adds line numbers to the given C program file and display the same in the standard output.

```
%{
int yylineno;
%}
%%
^(.*)\n printf("%4d\t%s", ++yylineno, yytext);
%%
int yywrap(void) {
return 1;
}
int main(int argc, char *argv[]) {
yyin = fopen(argv[1], "r");
yylex();
fclose(yyin);
}
```

Output

```
C:\Users\Shivaji V\LexPrograms>lines.exe sample.html
   2    <html>
   3    <body>
   4    <h1>My First Heading</h1>
   5    <p>My first paragraph.</p>
   6    </body>
```

Q26) Write a LEX program to count the number of comment lines in a given C program and eliminate them and write into another file.

```
%{
int com=0;
%}
%s COMMENT
%%
"/*" {BEGIN COMMENT;}
<COMMENT>"*/" {BEGIN 0; com++;}
<COMMENT>\n {com++;}
<COMMENT>. {;}
\/\/.* {; com++;}
.|\n {fprintf(yyout,"%s",yytext);}
%%
void main(int argc, char *argv[])
{
if(argc!=3)
{
printf("usage : a.exe input.c output.c\n");
exit(0);
}
yyin=fopen(argv[1],"r");
yyout=fopen(argv[2],"w");
yylex();
printf("\n number of comments are = %d\n",com);
}
int yywrap()
{
return 1;
}
```

Output

```
C:\Users\Shivaji V\LexPrograms>comments.exe input.c
usage : a.exe input.c output.c

C:\Users\Shivaji V\LexPrograms>comments.exe input.c output.c

 number of comments are = 2
```

Q27) Write a LEX program to identify the capital words from the given input.

```
%%
[A-Z]+        { printf("%s is a capital word\n", yytext); }
.|\n|\t| " "   { }
%%

int main() {
   printf("Enter String:\n");
   yylex();
   return 0;
}
int yywrap() { return 1; }
```

Output

```
Shivaji
S is a capital word
SaveethA uniVERsitY
S is a capital word
A is a capital word
VER is a capital word
Y is a capital word
```

Q28) Write a LEX Program to check the email address is valid or not.

```
%{
#include <stdio.h>
int valid = 0;
%}
%%
^[a-zA-Z0-9_.]+@[a-zA-Z0-9_.]+\.[a-zA-Z]+$ {
   valid = 1;
   return 0;
}
.|\n { }
%%
int yywrap() { return 1; }
int main() // Removed (argc, argv)
{
   // yyin is implicitly set to stdin (keyboard)
   yylex();
   if(valid)
      printf("Valid Email\n");
   else
      printf("Invalid Email\n");
   return 0;
}
```

Output

```
Shivaji.15@gmail.com
Valid Email

C:\Users\Shivaji V\LexPrograms>email.exe
12qwert.gmail,kom
Invalid Email
```

Q29) Write a LEX Program to convert the substring abc to ABC from the given input string

```
%{
int i;
%}
%%
[a-z A-Z]* { for(i=0;i<=yyleng;i++)
 { if((yytext[i]=='a')&&(yytext[i+1]=='b')&&(yytext[i+2]=='c'))
 { yytext[i]='A';
 yytext[i+1]='B';
 yytext[i+2]='C';
 }
 }
 printf("%s",yytext);
 }
[\t]* return 1;
.* {ECHO;}
\n {printf("%s",yytext);}
%%
int main()
{
yylex();
}
int yywrap()
{
return 1;
}
```

Output

```
abc is not abcedfgh
ABC is not ABCedfgh
bcz abc165adklhf
bcz abc165adklhf
```

Q30) Implement a LEX program to check whether the mobile number is valid or not.

```
%%
[0-9]+    { printf("\nValid digit\n"); }
.*        { printf("\nInvalid digit\n"); }
%%
int yywrap() { return 1; }
int main() {
   yylex();
   return 0;
}
```

Output

```
Enter Mobile Number : 3265147892031

Mobile Number Invalid

6549873210

Mobile Number Valid
```

Q31) Implement Lexical Analyzer using FLEX (Fast Lexical Analyzer). The program should separate the tokens in the given C program and display with appropriate caption.

```
digit [0-9]
letter [A-Za-z]
%{
int count_id,count_key;
%}
%%
(stdio.h|conio.h) { printf("%s is a standard library\n",yytext); }
(include|void|main|printf|int) { printf("%s is a keyword\n",yytext); count_key++; }
{letter}({letter}|{digit})* { printf("%s is a identifier\n", yytext); count_id++; }
{digit}+ { printf("%s is a number\n", yytext); }
\"(\\.|[^"\\])*\" { printf("%s is a string literal\n", yytext); }
.|\n { }
%%
int yywrap(void) {
return 1;
}
int main(int argc, char *argv[]) {
yyin = fopen(argv[1], "r");
yylex();
printf("number of identifiers = %d\n", count_id);
printf("number of keywords = %d\n", count_key);
fclose(yyin);
}
```

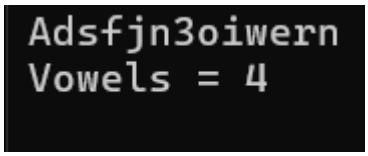
Output

```
C:\Users\Shivaji V\LexPrograms>flex.exe input.c
include is a keyword
stdio.h is a standard library
int is a keyword
main is a keyword
int is a keyword
a is a identifier
b is a identifier
c is a identifier
varible is a identifier
declaration is a identifier
printf is a keyword
enter is a identifier
two is a identifier
numbers is a identifier
scanf is a identifier
d is a identifier
d is a identifier
a is a identifier
b is a identifier
c is a identifier
a is a identifier
b is a identifier
adding is a identifier
two is a identifier
numbers is a identifier
printf is a keyword
sum is a identifier
is is a identifier
d is a identifier
c is a identifier
return is a identifier
0 is a number
number of identifiers = 24
number of keywords = 6
```

Q32) Write a LEX program to count the number of vowels in the given sentence.

```
%{
#include <stdio.h>
int v = 0;
%}
%%
[aeiouAEIOU]   { v++; }
.|\n          { }
%%
int yywrap() { return 1; }
int main(int argc, char *argv[]) {
   yyin = fopen(argv[1], "r");
   yylex();
   printf("Vowels = %d\n", v);
   return 0; }
```

Output



Q33) Write a LEX program to separate the keywords and identifiers.

```
digit [0-9]
letter [A-Za-z]
%{
int count_id,count_key;
%}
%%
(stdio.h|conio.h) { printf("%s is a standard library\n",yytext); }
(include|void|main|printf|int) { printf("%s is a keyword\n",yytext); count_key++; }
{letter}({letter}|{digit})* { printf("%s is a identifier\n", yytext); count_id++; }
{digit}+ { printf("%s is a number\n", yytext); }
\"(\\.|[^"\\])*\" { printf("%s is a string literal\n", yytext); }
.|\n { }
%%
int yywrap(void) {
return 1;
}
int main(int argc, char *argv[]) {
yyin = fopen(argv[1], "r");
yylex();
printf("number of identifiers = %d\n", count_id);
printf("number of keywords = %d\n", count_key);
fclose(yyin);
}
```

Output

```
C:\Users\Shivaji V\LexPrograms>flex.exe input.c
include is a keyword
stdio.h is a standard library
int is a keyword
main is a keyword
int is a keyword
a is a identifier
b is a identifier
c is a identifier
varible is a identifier
declaration is a identifier
printf is a keyword
enter is a identifier
two is a identifier
numbers is a identifier
scanf is a identifier
d is a identifier
d is a identifier
a is a identifier
b is a identifier
c is a identifier
a is a identifier
b is a identifier
adding is a identifier
two is a identifier
numbers is a identifier
printf is a keyword
sum is a identifier
is is a identifier
d is a identifier
c is a identifier
return is a identifier
0 is a number
number of identifiers = 24
number of keywords = 6
```

Q34) Write a LEX program to recognise numbers and words in a statement

```
%%
[\t ]+ ;
[0-9]+|[0-9]*\.[0-9]+ { printf("\n%s is NUMBER", yytext);}
#.* { printf("\n%s is COMMENT", yytext);}
[a-zA-Z]+ { printf("\n%s is WORD", yytext);}
\n { ECHO;}
%%
int main()
{
while( yylex());
}
int yywrap( )
{
return 1;
}
```

Output

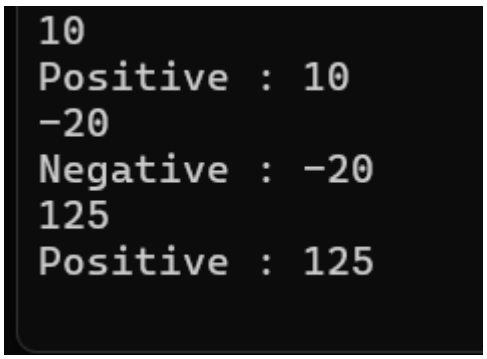
```
Alladin is 100 years OLD

Alladin is WORD
is is WORD
100 is NUMBER
years is WORD
OLD is WORD
```

Q35) Write a LEX program to identify and count positive and negative numbers.

```
%{
#include <stdio.h>
int pos = 0, neg = 0;
%}
%%
-[0-9]+      { neg++; printf("Negative : %s\n", yytext); }
\+?[0-9]+    { pos++; printf("Positive : %s\n", yytext); }
.|\n         { }
%%
int yywrap() { return 1; }
int main(int argc, char *argv[]) {
    yyin = fopen(argv[1], "r");
    yylex();
    printf("\nTotal Positive Numbers = %d\n", pos);
    printf("Total Negative Numbers = %d\n", neg);
    return 0;
}
```

Output



Q36) Write a LEX program to validate the URL.

```
%%
((http)|(ftp))s?:\/\/[a-zA-Z0-9](.[a-z])+(.[a-zA-Z0-9+=?]*)* {printf("\nURL Valid\n");}
.+ {printf("\nURL Invalid\n");}
%%
void main()
{
printf("\nEnter URL : ");
yylex();
printf("\n");
}
int yywrap()
{
}
```

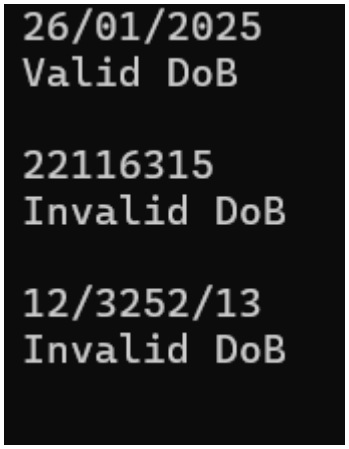
Output

Q37) Write a LEX program to validate DOB of students

```
%%
((0[1-9])|([12][0-9])|(3[01]))\/((0[1-9])|(1[0-2]))\/(19[0-9]{2}|2[0-9]{3})   { printf("Valid DoB\n"); }
.*                                                  { printf("Invalid DoB\n"); }
%%

int main() {
   yylex();
   return 0;
}
int yywrap() { return 1; }
```

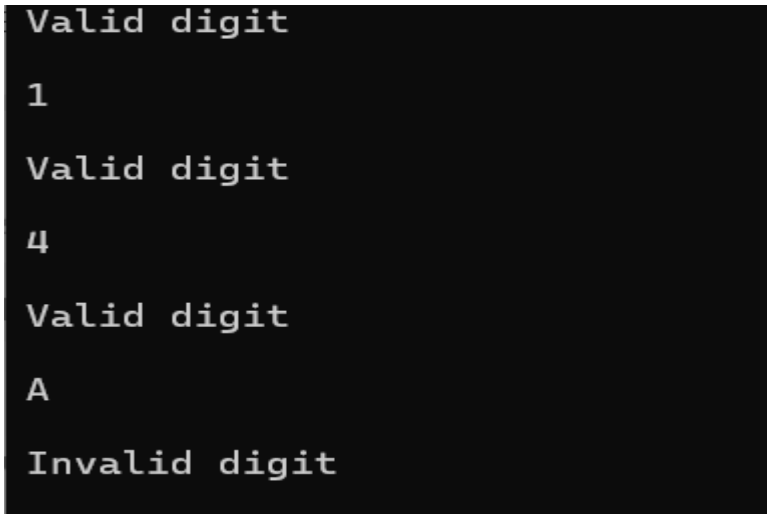
Output



Q38) Write a LEX program to check whether the given input is digit or not.

```
%%
[0-9]+    { printf("\nValid digit\n"); }
.*        { printf("\nInvalid digit\n"); }
%%
int yywrap() { return 1; }
int main() {
   yylex();
   return 0;
}
```

Output



Q39) Write a LEX program to implement basic mathematical operations.

```
%{
#include <stdio.h>
int a, b;
%}
%%
([0-9]+)\+([0-9]+)    { sscanf(yytext, "%d+%d", &a, &b); printf("Result = %d\n", a + b); }
([0-9]+)\-([0-9]+)    { sscanf(yytext, "%d-%d", &a, &b); printf("Result = %d\n", a - b); }
([0-9]+)\*([0-9]+)    { sscanf(yytext, "%d*%d", &a, &b); printf("Result = %d\n", a * b); }
([0-9]+)/([0-9]+)     { sscanf(yytext, "%d/%d", &a, &b);
                        if(b == 0) printf("Cannot divide by zero\n");
                        else printf("Result = %d\n", a / b);
                      }
.*                    { }
%%
int yywrap() { return 1; }
int main() {
   yylex();
   return 0;
}
```

Output

```
20+23
Result = 43

25*1
Result = 25

50-12
Result = 38

32/5
Result = 6
```