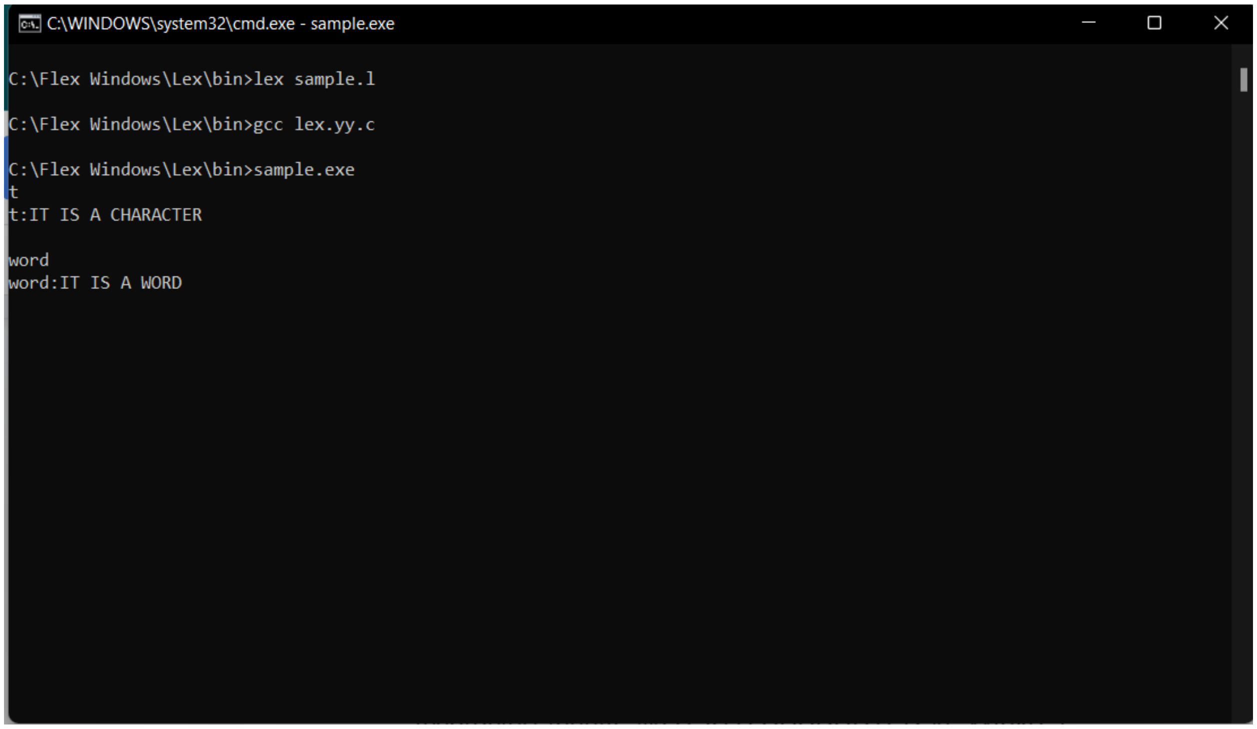
Prog 1. Write a lex program in flex editor to identify character and words.

```
%option noyywrap


%{
#include<stdio.h>
char character;
char word;
%}


word [a-zA-Z]+
character [A-Za-z]




%%
{character} {printf("%s:IT IS A CHARACTER \n",yytext); }
{word} {printf("%s:IT IS A WORD \n",yytext); }


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



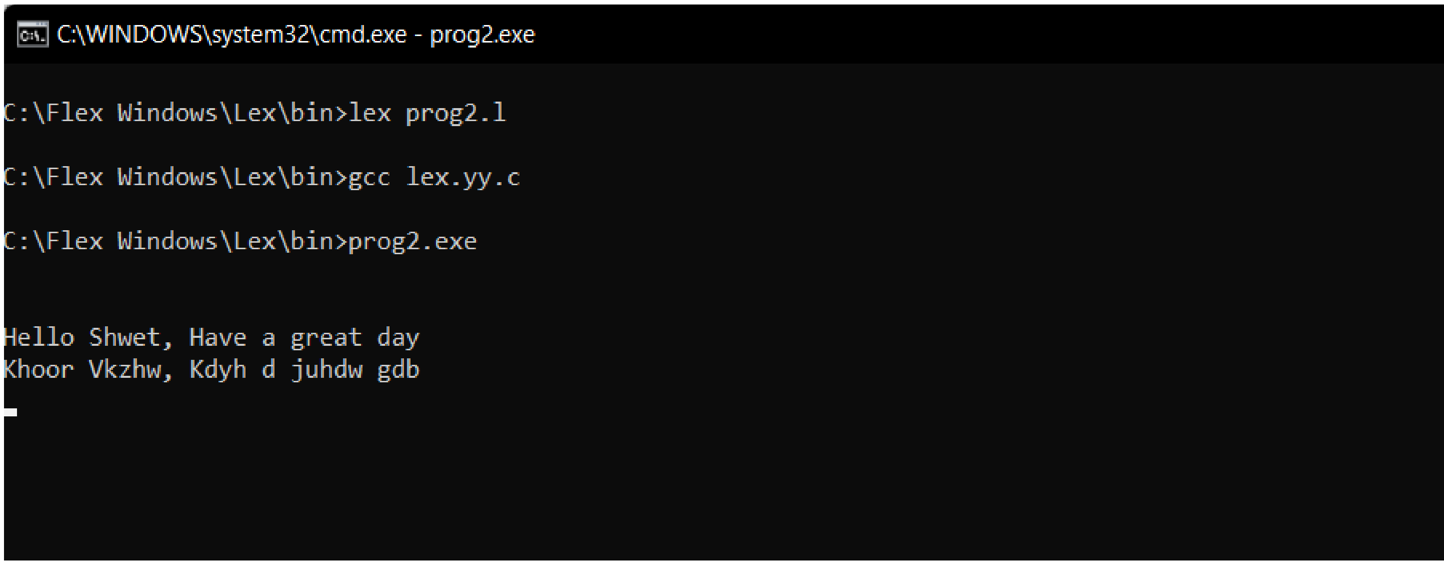
C:\WINDOWS\system32\cmd.exe - sample.exe

```
C:\Flex Windows\Lex\bin>lex sample.l

C:\Flex Windows\Lex\bin>gcc lex.yy.c

C:\Flex Windows\Lex\bin>sample.exe
t
t:IT IS A CHARACTER

word
word:IT IS A WORD
```

Ques2. Write a Lex program that implements the Caesar cipher: it replaces every letter with the one three letters after in alphabetical order, wrapping around at Z. e.g. a is replaced by d, b by e, and so on z by c.

Solution

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

%%
[A-Wa-w] {printf("%c", yytext[0]+3);}
[X-Zx-z] {printf("%c", yytext[0]-23);}

%%
int main()
{
     yylex();
     return 1;
}
```

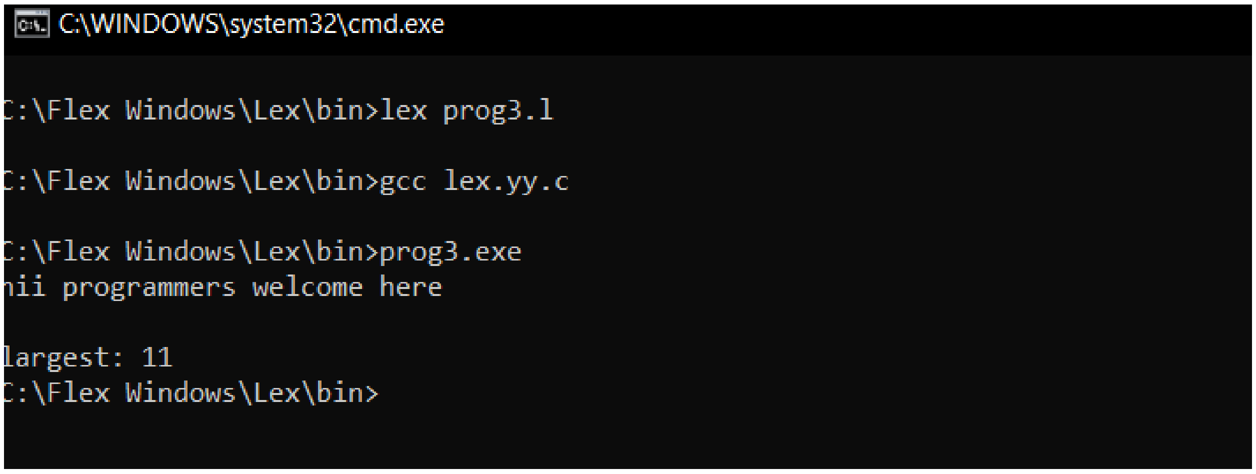


```
C:\Flex Windows\Lex\bin>lex prog2.l

C:\Flex Windows\Lex\bin>gcc lex.yy.c

C:\Flex Windows\Lex\bin>prog2.exe


Hello Shwet, Have a great day
Khoor Vkzhw, Kdyh d juhdw gdb
```

Practical 3

Ques. Write a Lex program that finds the longest word (defined as a contiguous string of upper-
and lower-case letters) in the input.

```
%option noyywrap
%{
	#include<stdio.h>
	int counter=0;
%}

%%
[a-zA-Z]+ {if(yyleng>counter)
			counter=yyleng;
	}
%%
int main()
{
	yylex();
	printf("largest: %d", counter);
	return 0;
}
```



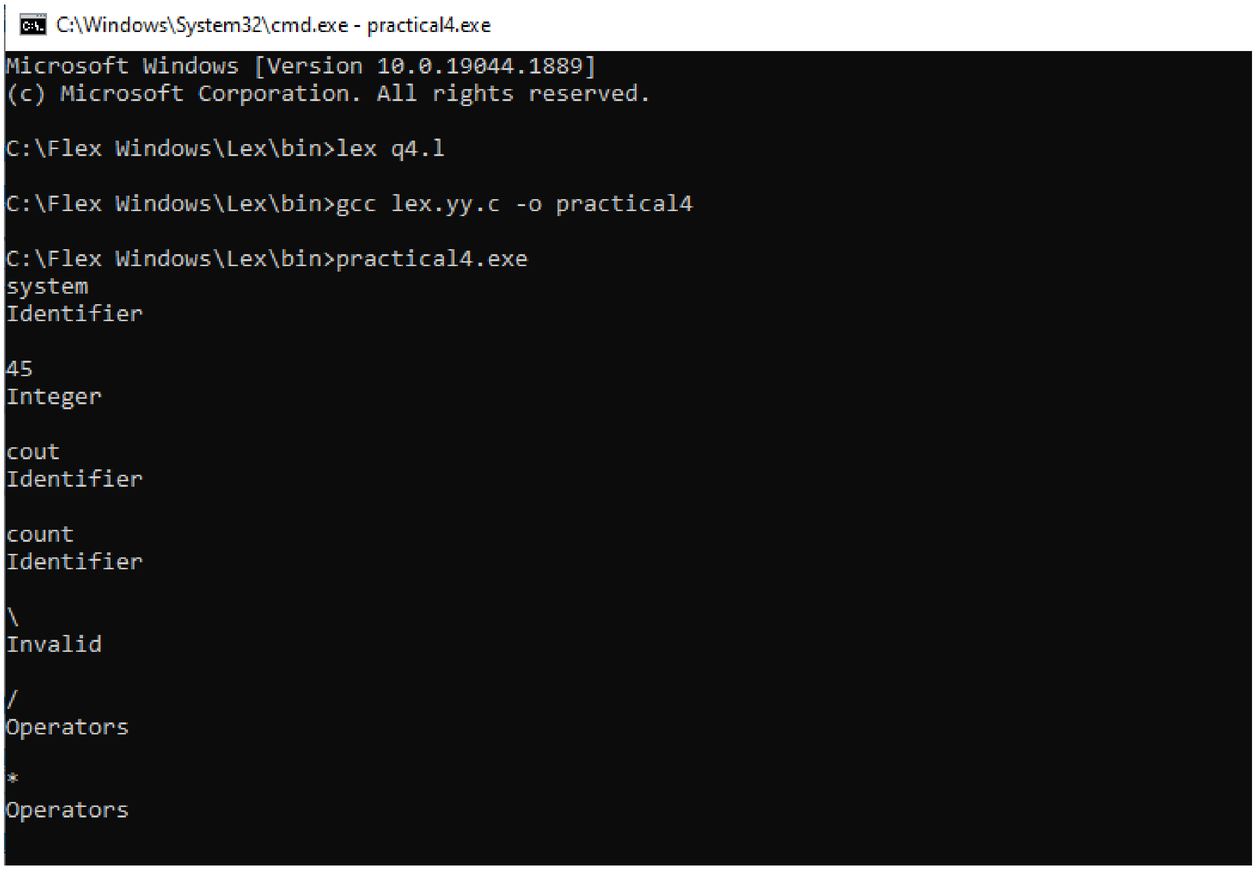
```
C:\WINDOWS\system32\cmd.exe

:\Flex Windows\Lex\bin>lex prog3.l

:\Flex Windows\Lex\bin>gcc lex.yy.c

:\Flex Windows\Lex\bin>prog3.exe
ii programmers welcome here

largest: 11
:\Flex Windows\Lex\bin>
```

PRACTICAL 4

Question 4 Write a Lex program that distinguishes keywords, integers, floats, identifiers, operators, and comments in any simple programming language.

```
%{

#include<stdio.h>

%}

%%

[0-9]* {printf("Integer\n");}

[0-9]+\.[0-9]+ {printf("Float\n"); }

int|float|if|else|printf|main|exit|switch {printf("Keyword\n");}

[+|*|/|%|&] {printf("Operators\n");}

"-" {printf("Operators\n");}

"/*".*"*/" {printf("comment\n");}

[_a-zA-Z][_a-zA-Z0-9]{0,30} {printf("Identifier\n");}

. {printf("Invalid\n");}

%%

int main()

{

yylex();

return 0;

}

int yywrap()

{

return 1;
}
```

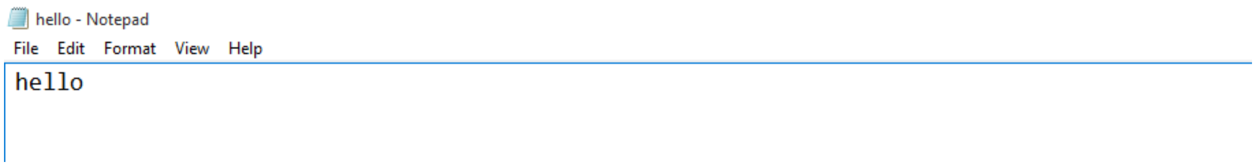
Output:



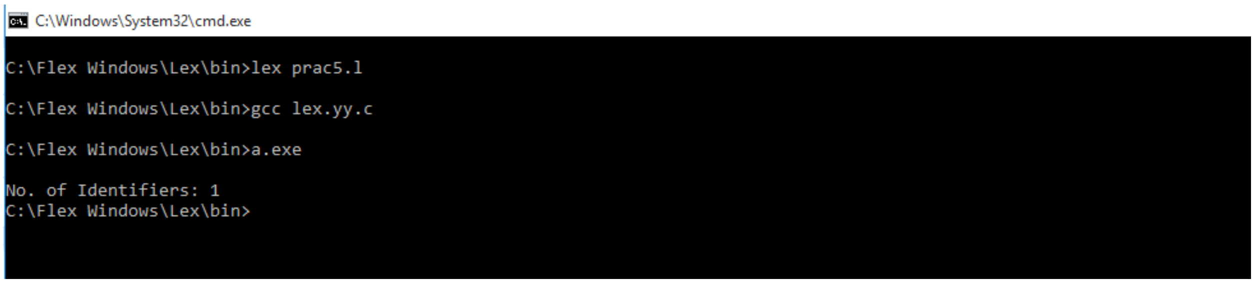
```
C:\Windows\System32\cmd.exe - practical4.exe
Microsoft Windows [Version 10.0.19044.1889]
(c) Microsoft Corporation. All rights reserved.

C:\Flex Windows\Lex\bin>lex q4.l

C:\Flex Windows\Lex\bin>gcc lex.yy.c -o practical4

C:\Flex Windows\Lex\bin>practical4.exe
system
Identifier

45
Integer

cout
Identifier

count
Identifier

\
Invalid

/
Operators

*
Operators
```

Ques 5. Write a lex program to count the number of identifiers in a C file.

```
%option noyywrap

%{
        #include<stdio.h>
        int count = 0;
%}
%%
[A-Z a-z _][A-Z a-z 0-9 _]* count++;

%%
int main(){
        yyin = fopen("hello.txt", "r");
        yylex();
        printf("\nNo. of Identifiers: %d", count);
        return 0;
}
```



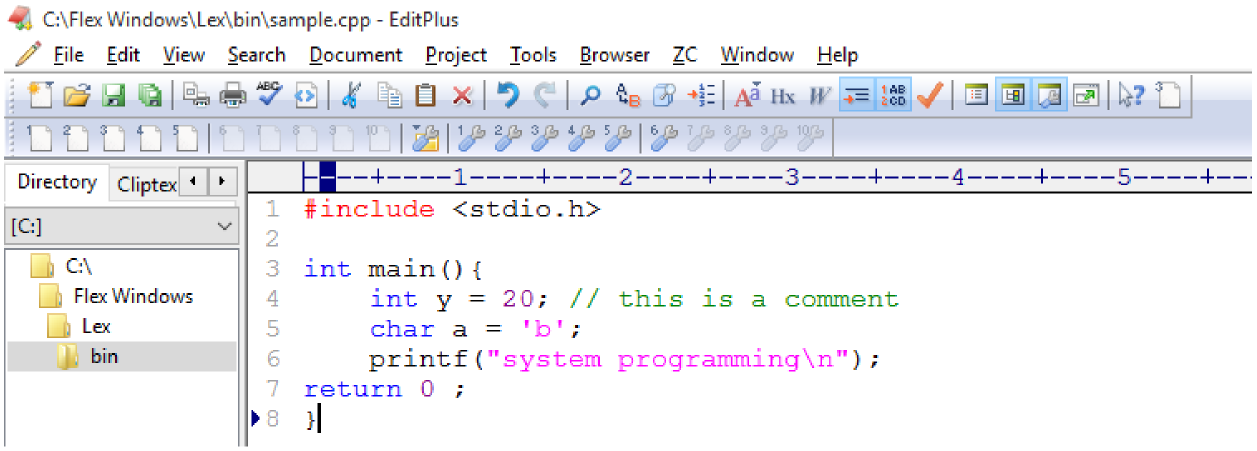
```
hello
```



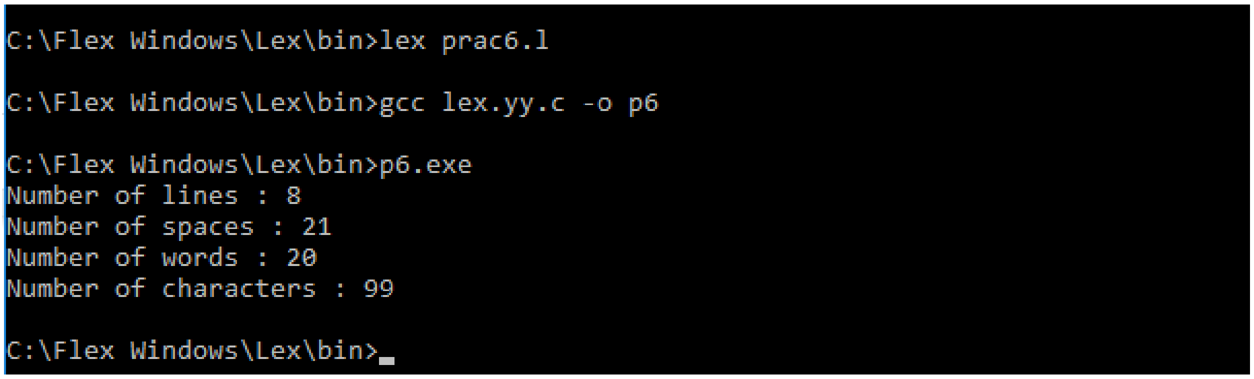
```
C:\Flex Windows\Lex\bin>lex prac5.l

C:\Flex Windows\Lex\bin>gcc lex.yy.c

C:\Flex Windows\Lex\bin>a.exe

No. of Identifiers: 1
C:\Flex Windows\Lex\bin>
```

Ques6. Write a lex program to count the number of words , characters, blank spaces and lines in a c/cpp file

```
%option noyywrap
%{
#include<stdio.h>
#include<string.h>
int lines = 0, nchar = 0, nspc = 0, nwrd = 0;
%}
%%
[\n]|[.] {lines++; }
[A-Za-z|0-9]+ {nwrd++;nchar = nchar+strlen(yytext);}
([ ])|[\t|\r]+ {nspc++;}
. {nchar++;}
%%
int main()
{
yyin=fopen("sample.cpp", "r");
yylex();
printf("Number of lines : %d\n", lines);
printf("Number of spaces : %d\n", nwrd);
printf("Number of words : %d\n", nspc);
printf("Number of characters : %d\n", nchar);
return 0;
}
```


C:\Flex Windows\Lex\bin\sample.cpp - EditPlus
File Edit View Search Document Project Tools Browser ZC Window Help
Directory
Cliptex
[C:]
C:\
Flex Windows
Lex
bin

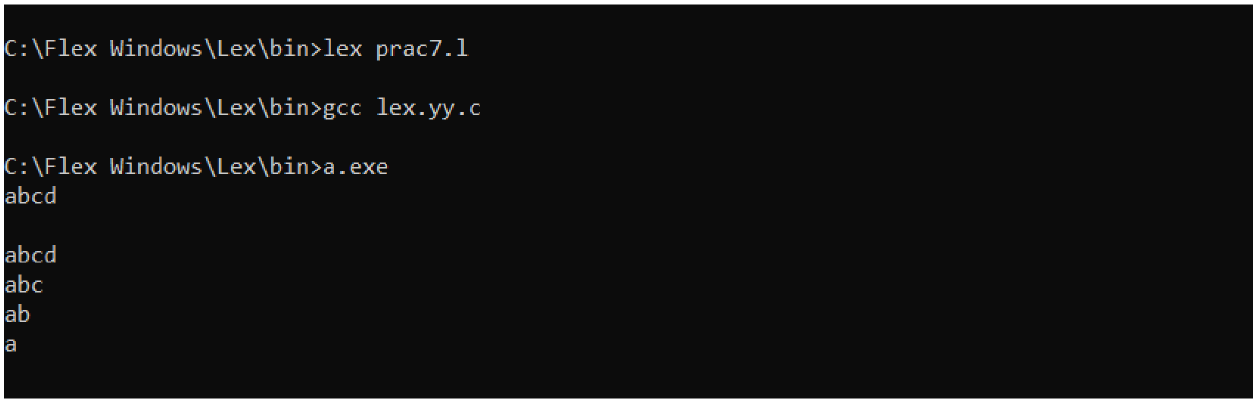
```
#include <stdio.h>

int main(){
    int y = 20; // this is a comment
    char a = 'b';
    printf("system programming\n");
return 0 ;
}
```



```
C:\Flex Windows\Lex\bin>lex prac6.l

C:\Flex Windows\Lex\bin>gcc lex.yy.c -o p6

C:\Flex Windows\Lex\bin>p6.exe
Number of lines : 8
Number of spaces : 21
Number of words : 20
Number of characters : 99

C:\Flex Windows\Lex\bin>
```

Ques. Write a Lex specification program that generates a C program which takes a string “abcd” and prints the following output.
abcd
abc
ab
a

```
%option noyywrap
%{
        #include<stdio.h>
        char ch;
        char i,j;
%}


%%
[a-zA-Z]* {printf("\n");
for(i=0; i<yyleng; i++)
{

        for(j = 0; j<yyleng-i; j++)
{
printf("%c",yytext[j]);
}
printf("\n");
}
}



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

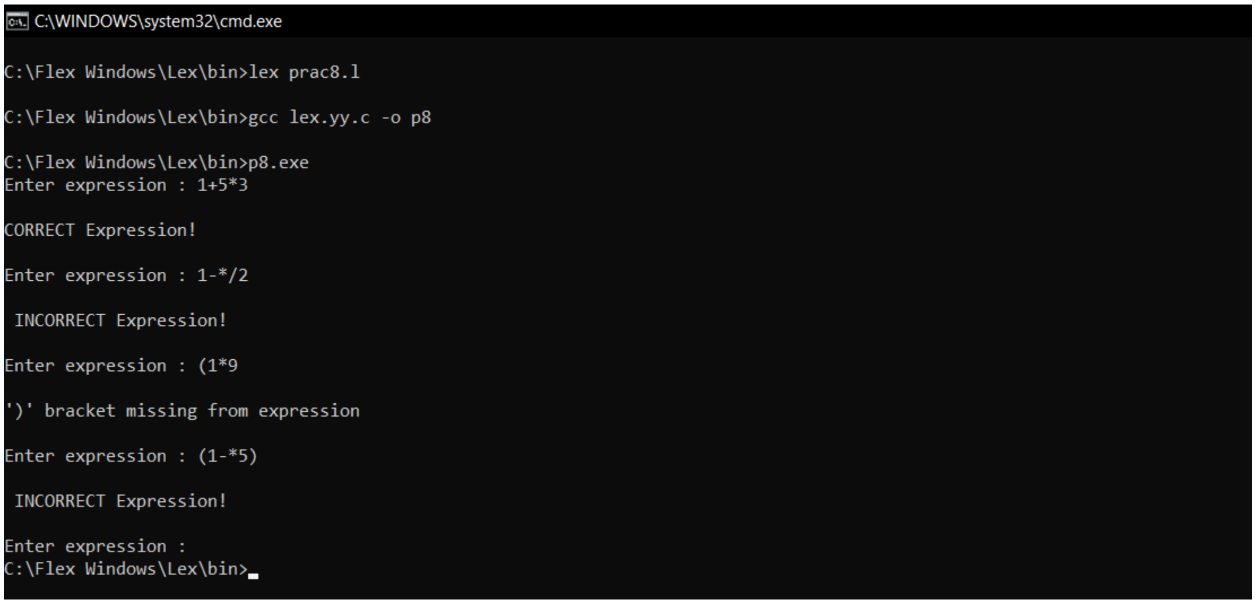


```
C:\Flex Windows\Lex\bin>lex prac7.l

C:\Flex Windows\Lex\bin>gcc lex.yy.c

C:\Flex Windows\Lex\bin>a.exe
abcd

abcd
abc
ab
a
```

Ques 8. Write a program in lex to recognize a valid arithmetic expression.

```
%option noyywrap
%{
#include<strings.h>
int opcount=0,intcount=0,check=1,top=0, prnt=0;;
%}
%%
['('] {check=0;}
[')'] {check=1;}
[+|*|/|-] {opcount++; prnt=1;}
[0-9]+ {intcount++; prnt=1;}
. {printf("Input is Invalid (only digits and +|-|*|/ is valid\n");}
[\n] {
if(prnt==1)
{
if(intcount==opcount+1)
{
if(check==1)
{
printf("\nCORRECT Expression!\n");
}
else{
printf("\n')' bracket missing from expression\n");
}
}
else
{
printf("\n INCORRECT Expression!\n");
}
prnt=0;
opcount=0;
intcount=0;
check=1;
printf("\nEnter expression : ");
}
else
{
printf("Please, Continue or terminate by(ctrl+c). ");
printf("\nEnter expression : ");
}
}
%%
int main()
```

```
{
printf("Enter expression : ");
yylex();
return 0;
}
```



```
C:\Flex Windows\Lex\bin>lex prac8.l

C:\Flex Windows\Lex\bin>gcc lex.yy.c -o p8

C:\Flex Windows\Lex\bin>p8.exe
Enter expression : 1+5*3

CORRECT Expression!

Enter expression : 1-*/2

 INCORRECT Expression!

Enter expression : (1*9

')' bracket missing from expression

Enter expression : (1-*5)

 INCORRECT Expression!

Enter expression :
C:\Flex Windows\Lex\bin>
```

PRACTICAL-9

Q5. Write a YACC program to find the validity of a given expression (for operators + - * and /)

CODE:

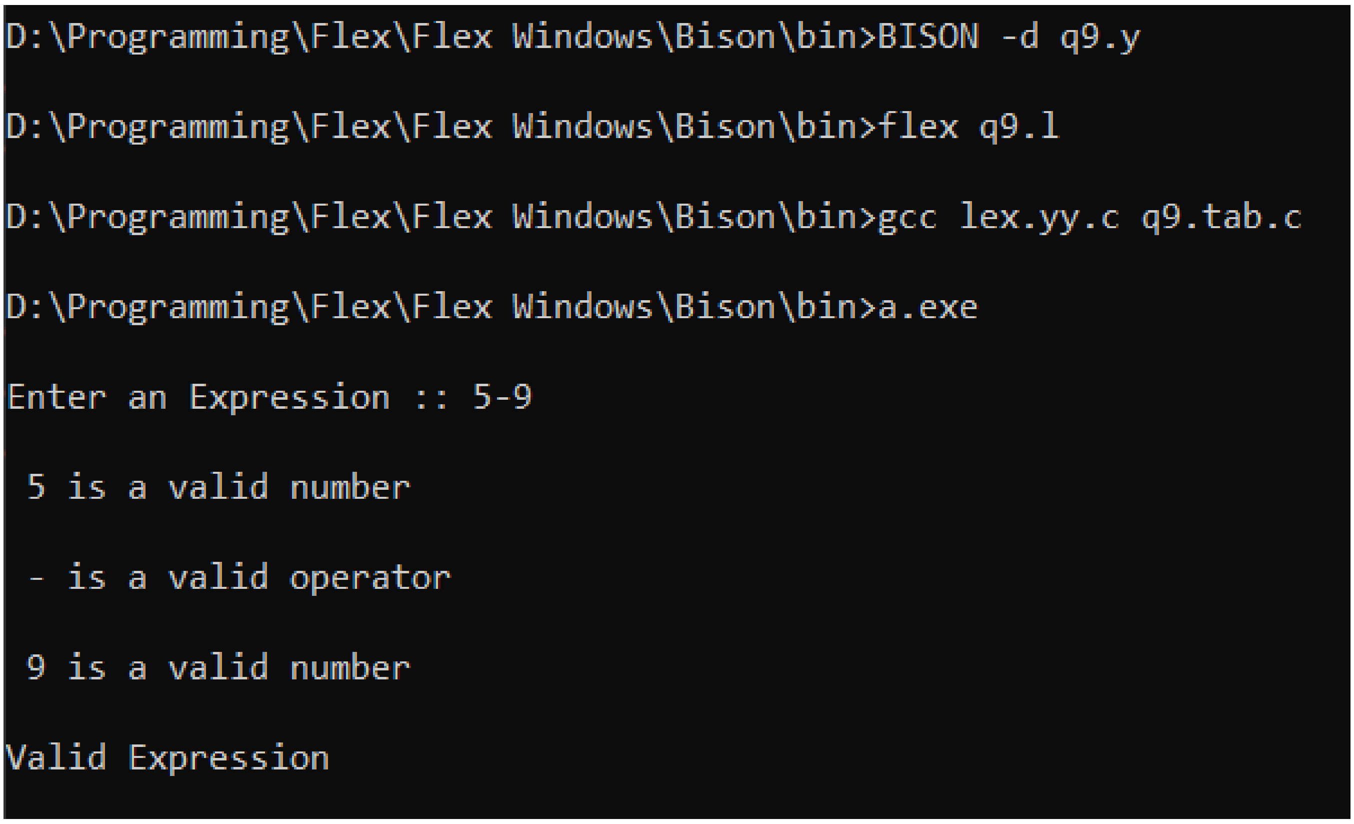
q9.l

```
%{
#include<stdio.h>
#include<stdlib.h>
#include "q9.tab.h"
%}
%%
[\t]+ ;
[0-9]+ { printf("\n %s is a valid number\n",yytext);
return NUM;}
[a-z_]+[a-z_0-9]* { printf("\n %s is a valid variable\n",yytext);
return VAR;}
[+] {printf("\n %s is a valid operator\n",yytext);
return '+';}
[-] {printf("\n %s is a valid operator\n",yytext);
return '-';}
[/] {printf("\n %s is a valid operator\n",yytext);
return '/';}

[*] {printf("\n %s is a valid operator\n",yytext);

return '*';}

\n {return NL;}

. {return yytext[0];}

%%
```

q9.y

```
%{

#include "q9.tab.h"

#include<stdio.h>

#include<stdlib.h>

%}

%token NUM VAR NL

%%


%left '+' '-' '*' '/' ;


S : S1 NL{printf("\nValid Expression\n");return 0;}

S1 : S1 '+' S1|S1 '-' S1|S1 '/' S1|S1 '*' S1| '(' S1 ')' | VAR | NUM |;

%%

int main(){

printf("\nEnter an Expression :: ");

yyparse();

return 0;

}
```

```
int yywrap(){}

int yyerror(){

printf("\nInvalid Expression\n");

exit(1);

}
```



```
D:\Programming\Flex\Flex Windows\Bison\bin>BISON -d q9.y

D:\Programming\Flex\Flex Windows\Bison\bin>flex q9.l

D:\Programming\Flex\Flex Windows\Bison\bin>gcc lex.yy.c q9.tab.c

D:\Programming\Flex\Flex Windows\Bison\bin>a.exe

Enter an Expression :: 5-9

 5 is a valid number

 - is a valid operator

 9 is a valid number

Valid Expression
```

PRACTICAL-10

Q10. A Program in YACC which recognizes a valid variable which starts with letter followed by a digit. The letter should be in lowercase only.

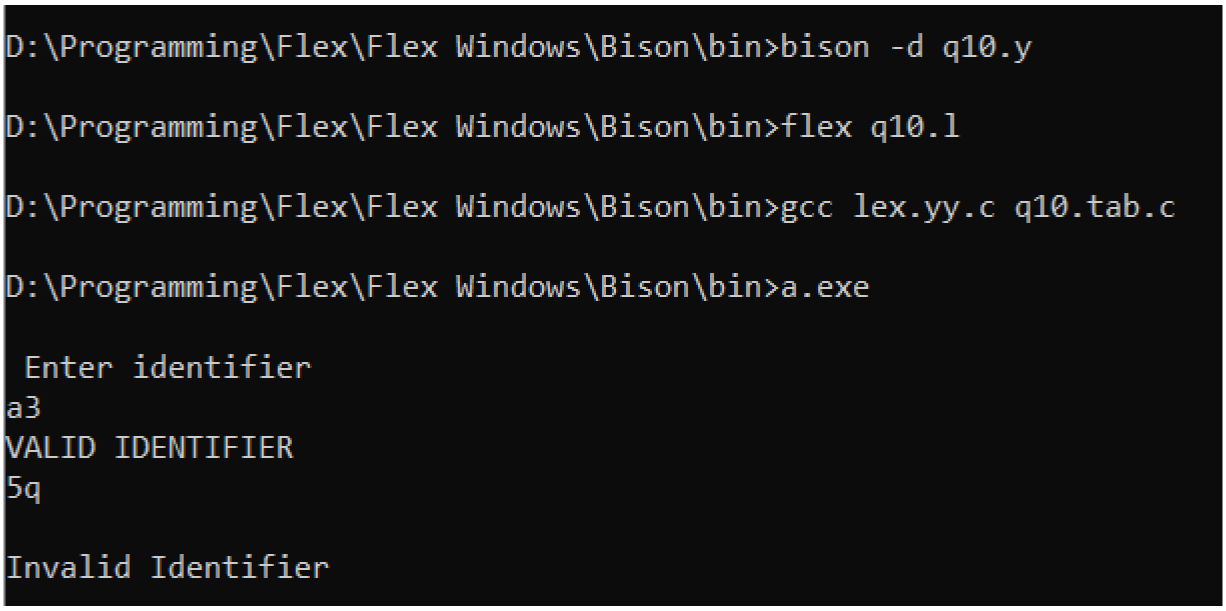
CODE:

Q10.l

```
%{
#include <stdio.h>
#include <stdlib.h>
#include "q10.tab.h"
%}
%option noyywrap
%%
[a-z] { return L; }
[0-9] { return D; }
[ \t\n]+ { ; }
. { return yytext[0]; }
%%
```

Q10.y

```
%{

#include <stdio.h>

#include <stdlib.h>

#include "q10.tab.h"

%}

%token  D L

%%

S : L D { printf("VALID IDENTIFIER\n"); }

;

%%

int main()

{

printf("\n Enter identifier\n");

yyparse();

return 0;

}

int yywrap(){}

int yyerror(){

printf("\nInvalid  Identifier\n");

exit(1);

}
```


```
D:\Programming\Flex\Flex Windows\Bison\bin>bison -d q10.y

D:\Programming\Flex\Flex Windows\Bison\bin>flex q10.l

D:\Programming\Flex\Flex Windows\Bison\bin>gcc lex.yy.c q10.tab.c

D:\Programming\Flex\Flex Windows\Bison\bin>a.exe

 Enter identifier
a3
VALID IDENTIFIER
5q

Invalid Identifier
```

PRACTICAL-11

Q11. A Program in YACC to evaluate an expression (simple calculator program for addition and subtraction, multiplication, division)

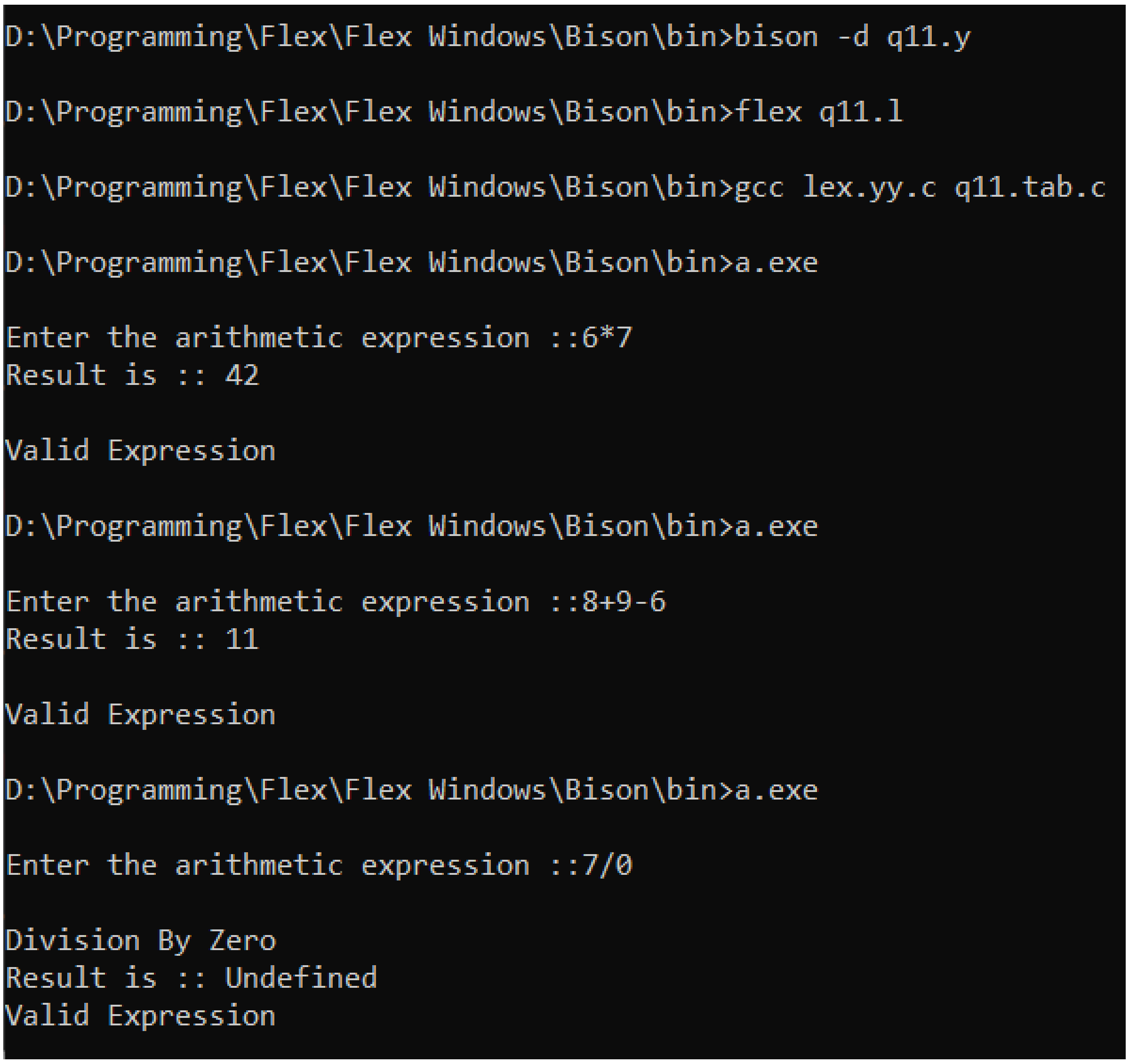
CODE:

Q11.l

```
%{

#include<stdio.h>

#include<stdlib.h>

#include "q11.tab.h"

int yylval;

%}

%%

[0-9]+ {yylval = atoi(yytext);

return NUM;}

[\t]+ ;

\n {return 0;}

. {return yytext[0];}

%%
```

Q11.y

```
%{
#include<stdio.h>
#include<stdlib.h>
#include "q11.tab.h"
%}
%token NUM
%left '+' '-'
%left '/' '*'
%left '(' ')'
%%
expr:e{printf("Result is :: %d\n",$$);
return 0;}
e:e '+' e{$$ = $1+$3;}
|e '-' e{$$ = $1-$3;}
|e '*' e{$$ = $1*$3;}
|e '/' e{
if($3==0){
printf("\nDivision By Zero\n");
printf("Result is :: Undefined");
return 0;
}
else
$$ = $1/$3;}

|'(' e ')'{$$ = $2;}

|NUM {$$ = $1;}

%%

int main(){


printf("\nEnter the arithmetic expression ::");

yyparse();

printf("\nValid Expression\n");

return 0;

}

int yywrap(){

return 0;

}

int yyerror(){

printf("\nInvalid Expression\n");

exit(1);

}
```



```
D:\Programming\Flex\Flex Windows\Bison\bin>bison -d q11.y

D:\Programming\Flex\Flex Windows\Bison\bin>flex q11.l

D:\Programming\Flex\Flex Windows\Bison\bin>gcc lex.yy.c q11.tab.c

D:\Programming\Flex\Flex Windows\Bison\bin>a.exe

Enter the arithmetic expression ::6*7
Result is :: 42

Valid Expression

D:\Programming\Flex\Flex Windows\Bison\bin>a.exe

Enter the arithmetic expression ::8+9-6
Result is :: 11

Valid Expression

D:\Programming\Flex\Flex Windows\Bison\bin>a.exe

Enter the arithmetic expression ::7/0

Division By Zero
Result is :: Undefined
Valid Expression
```

PRACTICAL-12

Q12. Program in YACC to recognize the strings “ab”, “aabb”, ”aaabbb”,... of the language (anbn,n>=1).

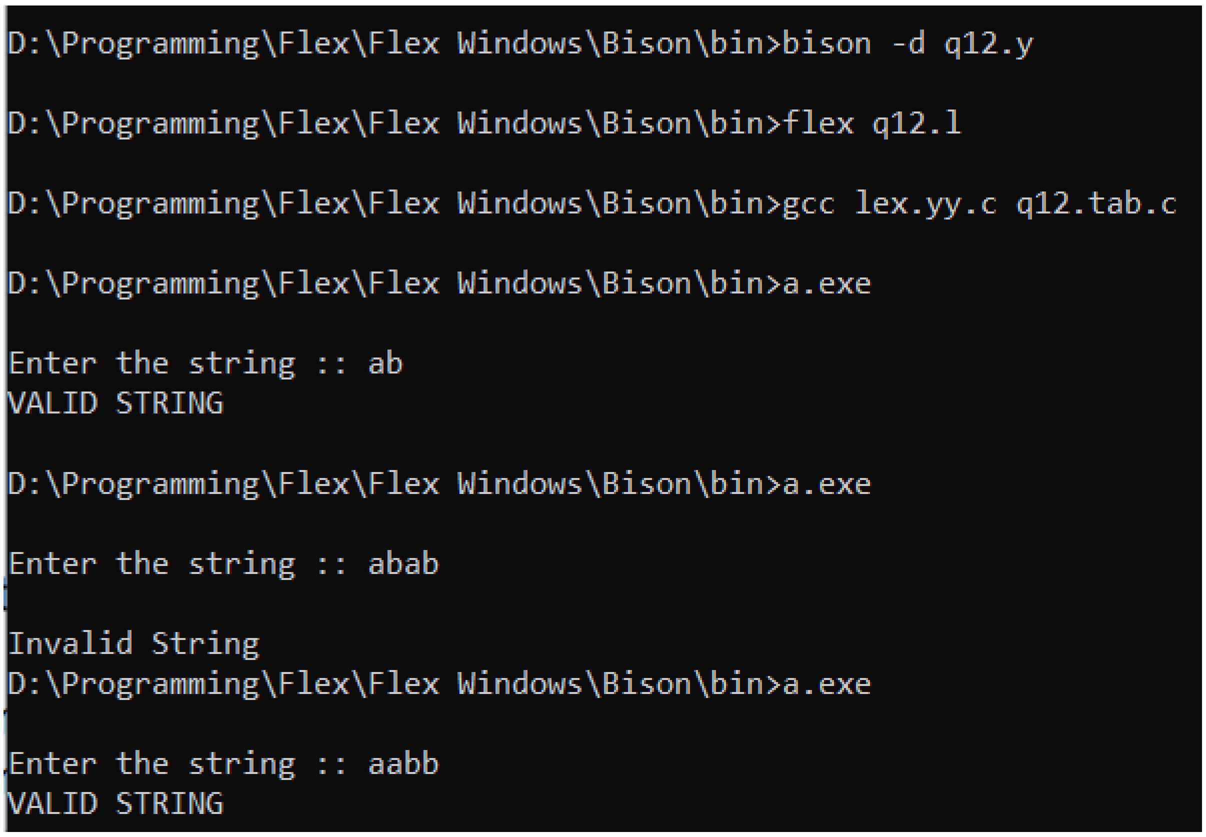
CODE:

Q12.l

```
%{

#include<stdio.h>

#include<stdlib.h>

#include "q12.tab.h"

%}

%option noyywrap

%%

[a] { return A; }

[b] { return B; }

[ |\n|\t] { return yytext[0]; }

. { return yytext[0]; }

%%
```

Q12.y

```
%{
#include<stdio.h>
#include<stdlib.h>
#include "q12.tab.h"
%}
%token A B
%%
S : E '\n' { printf("VALID STRING\n"); exit(0); }
;

E : A E B
| A B
;
%%
int main(){
printf("\nEnter the string :: ");
yyparse();
return 0;
}
yywrap(){}
yyerror(){
printf("\nInvalid String");
}
```



```
D:\Programming\Flex\Flex Windows\Bison\bin>bison -d q12.y

D:\Programming\Flex\Flex Windows\Bison\bin>flex q12.l

D:\Programming\Flex\Flex Windows\Bison\bin>gcc lex.yy.c q12.tab.c

D:\Programming\Flex\Flex Windows\Bison\bin>a.exe

Enter the string :: ab
VALID STRING

D:\Programming\Flex\Flex Windows\Bison\bin>a.exe

Enter the string :: abab

Invalid String
D:\Programming\Flex\Flex Windows\Bison\bin>a.exe

Enter the string :: aabb
VALID STRING
```

PRACTICAL-13

Q13. Program in YACC to recognize the language (anb , n>=10). (Output to say input is valid or not)

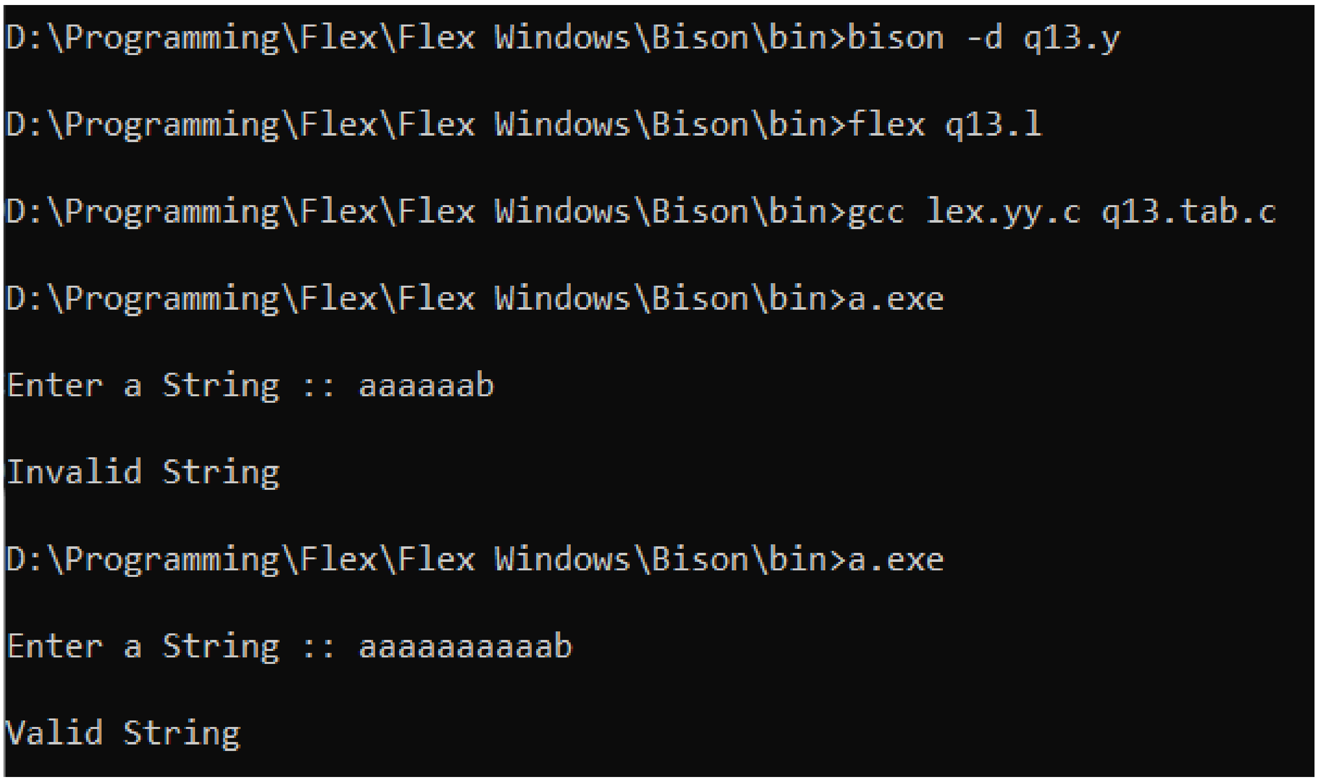
CODE:

Q13.l

```
%{
#include<stdio.h>
#include<stdlib.h>
#include "q13.tab.h"
%}
%%
[a] {return A;}
[b] {return B;}
\n {return NL;}
. {return yytext[0];}
%%
```

Q13.y

```
%{

#include<stdio.h>

#include<stdlib.h>

#include "q13.tab.h"

%}

%token A B NL

%%

S : A A A A A A A A A A S1 B NL

{ printf("\nValid String \n");

return 0;}


S1 : A S1

|;

%%

main(){

printf("\nEnter a String :: ");

yyparse();

}

yywrap(){}

yyerror(){

printf("\nInvalid String\n");

return 0;

}
```


```
D:\Programming\Flex\Flex Windows\Bison\bin>bison -d q13.y

D:\Programming\Flex\Flex Windows\Bison\bin>flex q13.l

D:\Programming\Flex\Flex Windows\Bison\bin>gcc lex.yy.c q13.tab.c

D:\Programming\Flex\Flex Windows\Bison\bin>a.exe

Enter a String :: aaaaaab

Invalid String

D:\Programming\Flex\Flex Windows\Bison\bin>a.exe

Enter a String :: aaaaaaaaaab

Valid String
```