1. **Write a LEX program to recognize valid *arithmetic expressions.* Identifiers in the expression could be only integers and operators could be + and *. Count the identifiers & operators present and print them separately.**

arithexpr.l

```
%{
#include<stdio.h>
int a=0,s=0,m=0,d=0,ob=0,cb=0;
int flaga=0, flags=0, flagm=0, flagd=0;
%}
id [a-zA-Z]+
%%
{id} {printf("\n %s is an identifier\n",yytext);}
[+] {a++;flaga=1;}
[-] {s++;flags=1;}
[*] {m++;flagm=1;}
[/] {d++;flagd=1;}
[(] {ob++;}
[)] {cb++;}
%%
int main()
{
printf("Enter the expression\n");
yylex();
if(ob-cb==0)
{
printf("Valid expression\n");
}
else
{
printf("Invalid expression");
}
printf("\nAdd=%d\nSub=%d\nMul=%d\nDiv=%d\n",a,s,m,d);
printf("Operators are: \n");
if(flaga)
printf("+\n");
if(flags)
printf("-\n");
if(flagm)
printf("*\n");
if(flagd)
printf("/\n");
return 0;
}
int yywrap()
{
return -1;
}
```

Commands :

```
gedit arithexpr.l
 lex arithexpr.l
```

```
cc lex.yy.c
./a.out
```
Output :
```
Enter the expression
(a+b)

 a is an identifier

 b is an identifier

Valid expression

Add=1
Sub=0
Mul=0
Div=0
Operators are:
+
Enter the expression
(a+b

 a is an identifier

 b is an identifier

Invalid expression
Add=1
Sub=0
Mul=0
Div=0
Operators are:
+
```
2. Write a LEX program to recognize whether a given sentence is simple or compound
Input : My name is XYZ Expected output : Simple

Simpcomp.l
```
%{
        #include<stdio.h>
        int flag=0;
%}

%%
and |
or |
but |
because |
if |
then |
nevertheless  { flag=1; }
. ;
\n  { return 0; }
%%
```

```
int main()
{
	printf("Enter the sentence:\n");
	yylex();
	if(flag==0)
		printf("Simple sentence\n");
	else
		printf("compound sentence\n");
}

int yywrap( )
{
	return 1;
}
```

Commands :

```
gedit Simpcomp.l
 lex Simpcomp.l
cc lex.yy.c
./a.out
```

Output :

```
Enter the sentence:
hello how are you
Simple sentence

Enter the sentence:
hello dear and what are u doing or some
compound sentence
```

3. Write a LEX program to eliminate *comment lines* in a *C* program and copy the resulting program into a separate file.

Comments .l

```
%{
	#include<stdio.h>
	int cl=0;
%}
%%
("/*"[^*/]*"*/")|"//"(.)* {++cl;fprintf(yyout," ");}
. {fprintf(yyout,yytext);}

%%
void main(int argc,char *argv[])
{
yyin = fopen(argv[1],"r");
yyout = fopen(argv[2],"w");
yylex();
printf("The number of comment lines are %d\n",cl);

}
```

```
int yywrap()
{
return -1;
}
```
in.c

```
#include<stdio.h>
main()
{
int a,b,c;
/*declaration*/
//hello
getch();
}
```

Commands :

```
gedit comments.l
 lex comments.l
cc lex.yy.c
./a.out in.c out.c
```

Output :

The number of comment lines are 2

```
cat out.c

#include<stdio.h>
main()
{
int a,b,c;


getch();
}
```

4. Write a LEX program to count the number of characters, words, spaces and lines in each input file.

wordcharcount.l

```
%{
int sc=0,wc=0,cc=0,lc=0;
%}
%%
[ ]+ {++sc;}
[a-zA-Z0-9]+ {++wc;cc=cc+yyleng;}
[\n]+ {lc++;}
. ;
%%
int main(int argc, char *argv[])
{
yyin=fopen(argv[1],"r");
yylex();
```

```
printf("spaces= %d\n",sc);
printf("characters= %d\n",cc);
printf("lines= %d\n",lc);
printf("words= %d\n",wc);
}
int yywrap()
{
return -1;
}
```

wordcharcount.txt

```
hello world
abc
123456
world
xyz
```

Commands :

```
gedit wordcharcount.l
 lex wordcharcount.l
cc lex.yy.c wordcharcount.txt
./a.out
```

Output :

```
spaces= 1
characters= 27
lines= 5
words= 6
```

5. Write YACC program to evaluate *arithmetic expression* involving operators: +, -, *, and /.

evalarith.l

```
%{
   /* Definition section*/
   #include "y.tab.h"
   extern yylval;
%}

%%
[0-9]+    {
            yylval = atoi(yytext);
            return NUMBER;
          }

[a-zA-Z]+    { return ID; }
[ \t]+        ;  /*For skipping whitespaces*/

\n            { return 0; }
.           { return yytext[0]; }
```

```
%%
```

evalarith.y

```
%{
   /* Definition section */
  #include <stdio.h>
%}

%token NUMBER ID
// setting the precedence
// and associativity of operators
%left '+' '-'
%left '*' '/'

/* Rule Section */
%%
E : T       {
              printf("Result = %d\n", $$);
              return 0;
            }

T :
   T '+' T { $$ = $1 + $3; }
   | T '-' T { $$ = $1 - $3; }
   | T '*' T { $$ = $1 * $3; }
   | T '/' T { $$ = $1 / $3; }
   | '-' NUMBER { $$ = -$2; }
   | '-' ID { $$ = -$2; }
   | '(' T ')' { $$ = $2; }
   | NUMBER { $$ = $1; }
   | ID { $$ = $1; };
%%

int main() {
   printf("Enter the expression\n");
   yyparse();
}

/* For printing error messages */
int yyerror(char* s) {
   printf("\nExpression is invalid\n");
}
```

Commands :

```
lex evalarith.l
yacc -d evalarith.y
cc lex.yy.c y.tab.c -ll
./a.out
```

Output :

```
Enter the expression
7*(5-3)/2
```

Result = 7

Enter the expression
(2+(3-5)
Expression is invalid

6. Write a program using LEX/YAAC tools to recognize valid *identifier, operators* and *keywords* in the given text (C program) file and print the same.

Keyiden.l

```
%{
#include <stdio.h>
#include "y.tab.h"
extern yylval;
%}
%%
[ \t] ;
[+|-|*|/|=|<|>] {printf("operator is %s\n",yytext);return OP;}
[0-9]+ {yylval = atoi(yytext); printf("numbers is %d\n",yylval); return DIGIT;}
int|char|bool|float|void|for|do|while|return {printf("keyword is%s\n",yytext);return KEY;}
[a-zA-Z0-9]+ {printf("identifier is %s\n",yytext);return ID;}
. ;
%%
```

Keyiden.y

```
%{
#include <stdio.h>
#include <stdlib.h>
int id=0, dig=0, key=0, op=0;
%}
%token DIGIT ID KEY OP
%%
input:
DIGIT input {  dig++; }
| ID input { id++; }
| KEY input { key++; }
| OP input {op++;}
| DIGIT { dig++; }
| ID { id++; }
| KEY { key++; }
| OP { op++;}
;
%%
#include <stdio.h>
extern int yylex();
extern int yyparse();
extern FILE *yyin; main() {
FILE *myfile = fopen("input1.c", "r");
if (!myfile)
{
printf("I can't open input1.c!");
return -1;
```

```
}
yyin = myfile;
do
{
yyparse();
}
while (!feof(yyin));
printf("numbers = %d\nKeywords = %d\nIdentifiers = %d\noperators =%d\n",dig, key,id, op);
}
void yyerror() {
printf("EEK, parse error! Message: ");
}
```

input1.c

```
void main()
{
char a ;
float a123 ;
char c ;
char b123 ;
 if(sum==10)
 printf("Equal");
 else
printf("Not Equal");
}
```

Commands :

```
lex keyiden.l
yacc -d keyiden.y
cc lex.yy.c y.tab.c -ll
./a.out
```

Output :

```
keyword isvoid
identifier is main
keyword ischar
identifier is a
keyword isfloat
identifier is a123
keyword ischar
identifier is c
keyword ischar
identifier is b123
identifier is if
identifier is sum
operator is =
operator is =
numbers is 10

identifier is printf
identifier is Equal
identifier is else
```

```
identifier is printf
identifier is Not
identifier is Equal

numbers = 1
Keywords = 5
Identifiers = 13
operators =2
```

7. Implement a program using LEX/YAAC tools to recognize a valid identifier in a given ‘C’ file.

identifiers.l

```
%{
#include <stdio.h>
#include "y.tab.h"
extern yylval;
%}
%%
[a-zA-Z0-9]+ {printf("identifier is %s\n",yytext);return ID;}
. ;
%%
```

identifiers.y

```
%{
#include <stdio.h>
#include <stdlib.h>
int id=0;
%}
%token ID
%%
input:
ID input {  id++; }
| ID { id++; }
;
%%
#include <stdio.h>
extern int yylex();
extern int yyparse();
extern FILE *yyin;
main()
{
FILE *myfile = fopen("input2.c", "r");
if (!myfile)
{
printf("I can't open input2.c!");
return -1;
}
yyin = myfile;
do
{
yyparse();
```

```
}
while (!feof(yyin));
printf("Identifiers = %d\n",id);
}
void yyerror() {
printf("EEK, parse error! Message: ");
}
```

input2.c

```
void main()
{
char a ;
float a123 ;
char c ;
char b123 ;
 if(sum==10)
 printf("Equal");
 else
printf("Not Equal");
}
```

Commands :

```
lex identifiers.l
yacc -d identifiers.y
cc lex.yy.c y.tab.c -ll
./a.out
```

Output:

```
identifier is void
identifier is main
identifier is char
identifier is a
identifier is float
identifier is a123
identifier is char
identifier is c
identifier is char
identifier is b123
identifier is if
identifier is sum
identifier is 10
identifier is printf
identifier is Equal
identifier is else
identifier is printf
identifier is Not
identifier is Equal


Identifiers = 19
```

8. Implement a program using LEX/YACC tool to recognize all strings ending with *b* preceded by *n a's* using the grammar $a^n b$ (note: input *n* value).

string.l

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

string.y

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

%token A B

%%
input:S {printf("Success\n");exit(0);}
S:A S1 B|B
S1:A S1
|
;
%%
extern FILE *yyin;
void main()
{
FILE *input;
char str[30], chr[2];
int n;
printf("Enter the value of n");
scanf("%d",&n);
for(int i=0;i<n;i++)
{
strcat(str,"a");
}
printf("enter the last character of the string\n");
scanf("%s",chr);
strcat(str,chr);
strcat(str,"\0");
input = fopen("input.txt","w");
fprintf(input,"%s",str);
fclose(input);
input = fopen("input.txt","r");
yyin = input;
yyparse();
```

```
}
int yyerror()
{
printf("Error");
exit(0);
}
int yywrap()
{
return -1;
}
```

input.txt
Empty File to be created

Commands :
lex string.l
yacc -d string.y
cc lex.yy.c y.tab.c -ll
./a.out

Enter the value of n 5
enter the last character of the string
b
Success

cat input.txt
aaaaab

Enter the value of n 4
enter the last character of the string
a
Error

cat input.txt
aaaaa