PROGRAM 9: **Design a DFA in LEX Code which accepts string containing even number of 'a' and even number of 'b' over input alphabet {a, b}.**

Code:

```
%{
#include<stdio.h>
%}
reg (aa|bb)*((ab|ba)(aa|bb)*(ab|ba)(aa|bb)*)*
%%
{reg} printf(%s is accepted",yytext);
.* printf("%s is not accepted",yytext);
%%
int yywrap(){}
int main(int argc, char *argv[]){
extern FILE *yyin;
yyin= fopen("Input.txt","r");
yylex();
return 0;
}
```

OUTPUT:

```
geu@geu-OptiPlex-320:~$ flex 1.l
geu@geu-OptiPlex-320:~$ cc lex.yy.c -lfl
geu@geu-OptiPlex-320:~$ ./a.out
aa is accepted
aabb is accepted
abab is accepted
aaab is not accepted
aaabba is accepted
geu@geu-OptiPlex-320:~$
```

```
Open                                                    Save
1 aa
2 aabb
3 abab
4 aaab
5 aaabba
```

PROGRAM 10: Design a DFA in LEX Code which accepts string containing third last element 'a' over input alphabet {a, b}.

Code:

```
%{
#include<stdio.h>
 %}
reg (a*b*)*a(aa|bb|ab|ba)
%%
{reg} printf("%s is accepted", yytext);
.* printf("%s is not accepted", yytext);
%%
int yywrap(){return 1;}
int main()
{
extern FILE *yyin;
yyin = fopen("Input2.txt","r");
yylex();
return 0;
}
```

OUTPUT:

```
geu@geu-OptiPlex-320:~$ flex 2.l
geu@geu-OptiPlex-320:~$ cc lex.yy.c -lfl
geu@geu-OptiPlex-320:~$ ./a.out
aaabaababb is accepted
ab is not accepted
aa is not accepted
aabb is accepted
aba is accepted
abababbabbbb is not accepted
bbb is not accepted

geu@geu-OptiPlex-320:~$
```

```
Open                                                    Save
1 aaabaababb
2 ab
3 aa
4 aabb
5 aba
6 abababbabbbb
7 bbb
8
Plain Text   Tab Width: 4   Ln 8, Col 1   INS
```

PROGRAM 11: Design a DFA in LEX Code to Identify and print Integer & Float Constants and Identifier.

Code:

```
%{
#include<stdio.h>
%}
%%
[+-]?[0-9]"."[0-9]+ printf("%s is Float Constants",yytext);
[+-]?[0-9]+ printf("%s is Integer Constants",yytext);
[a-zA-Z]+|[a-zA-Z]+[0-9]+ printf("%s is Identifiers",yytext);
.* printf("%s is Neither Ineger, Float constants nor identifiers",yytext);
%%
int yywrap(){}
int main()
{
extern FILE *yyin;
yyin = fopen("Input3.txt","r");
yylex();
return 0;
}
```

OUTPUT:

```
geu@geu-OptiPlex-320:~$ flex 3.l
geu@geu-OptiPlex-320:~$ cc lex.yy.c -lfl
geu@geu-OptiPlex-320:~$ ./a.out
aab is Identifiers
90 is Integer Constants
+87 is Integer Constants
-0.78 is Float Constants
8.97 is Float Constants
1as is Neither Ineger,Float constants nor identifiers
a4566 is Identifiers
geu@geu-OptiPlex-320:~$
```

```
Open                                                    Save
1 aab
2 90
3 +87
4 -0.78
5 8.97
6 1as
7 a4566
Loading file '/home/geu/Input3.txt'...   Plain Text   Tab Width: 4   Ln 1, Col 1   INS
```

PROGRAM 12: Design YACC/LEX code to recognize valid arithmetic expression with operators +, -, * and /.

Code:

```
%{
#include <stdio.h>
#include <string.h>
int operators_count = 0, operands_count = 0, valid = 1, top = -1, l = 0, j = 0;
char operands[10][10], operators[10][10], stack[100];
%}

%%
"(" {top++; stack[top] = '(';}
"{" {top++; stack[top] = '{';}
"[" {top++; stack[top] = '[';}
")" {if (stack[top] != '('){
        valid = 0;
    }else if(operands_count>0 && (operands_count-operators_count)!=1){
        valid=0;
    }else{
        top--; operands_count=1; operators_count=0;
    }
}
"}" {
    if(stack[top] != '{') { valid = 0;
    }else if(operands_count>0 && (operands_count-operators_count)!=1){
        valid=0;
    }else{
        top--;
        operands_count=1;
        operators_count=0;
    }
}
"]" {if (stack[top] != '[') {
        valid = 0;
    }else if(operands_count>0 && (operands_count-operators_count)!=1){
        valid=0;
    }else{
        top--;
        operands_count=1;
        operators_count=0;
    }
}
"+"|"-"|"*"|"/" {
    operators_count++;
    strcpy(operators[l], yytext);
    l++;
}
[0-9]+|[a-zA-Z][a-zA-Z0-9_]* {
```

```
    operands_count++;
    strcpy(operands[j], yytext);
    j++;
}
%%

int yywrap(){return 1;}

int main()
{
    int k;
    printf("Enter the arithmetic expression: ");
    yylex();
    if (valid == 1 && top == -1) {
        printf("\nValid Expression\n");
    }else
        printf("\nInvalid Expression\n");
    return 0;
}
```

Output:

```
kirito@ubuntu:~/Documents/flex/file$ lex 12.l
kirito@ubuntu:~/Documents/flex/file$ gcc lex.yy.c -lfl
kirito@ubuntu:~/Documents/flex/file$ ./a.out
Enter the arithmetic expression: a+b


Valid Expression
kirito@ubuntu:~/Documents/flex/file$ ./a.out
Enter the arithmetic expression: (a-


Invalid Expression
kirito@ubuntu:~/Documents/flex/file$ ./a.out
Enter the arithmetic expression: (a*b-c)


Valid Expression
```

PROGRAM 13: Design YACC/LEX code to evaluate arithmetic expression involving operators +, -, * and / without operator precedence grammar.

Lex Program:

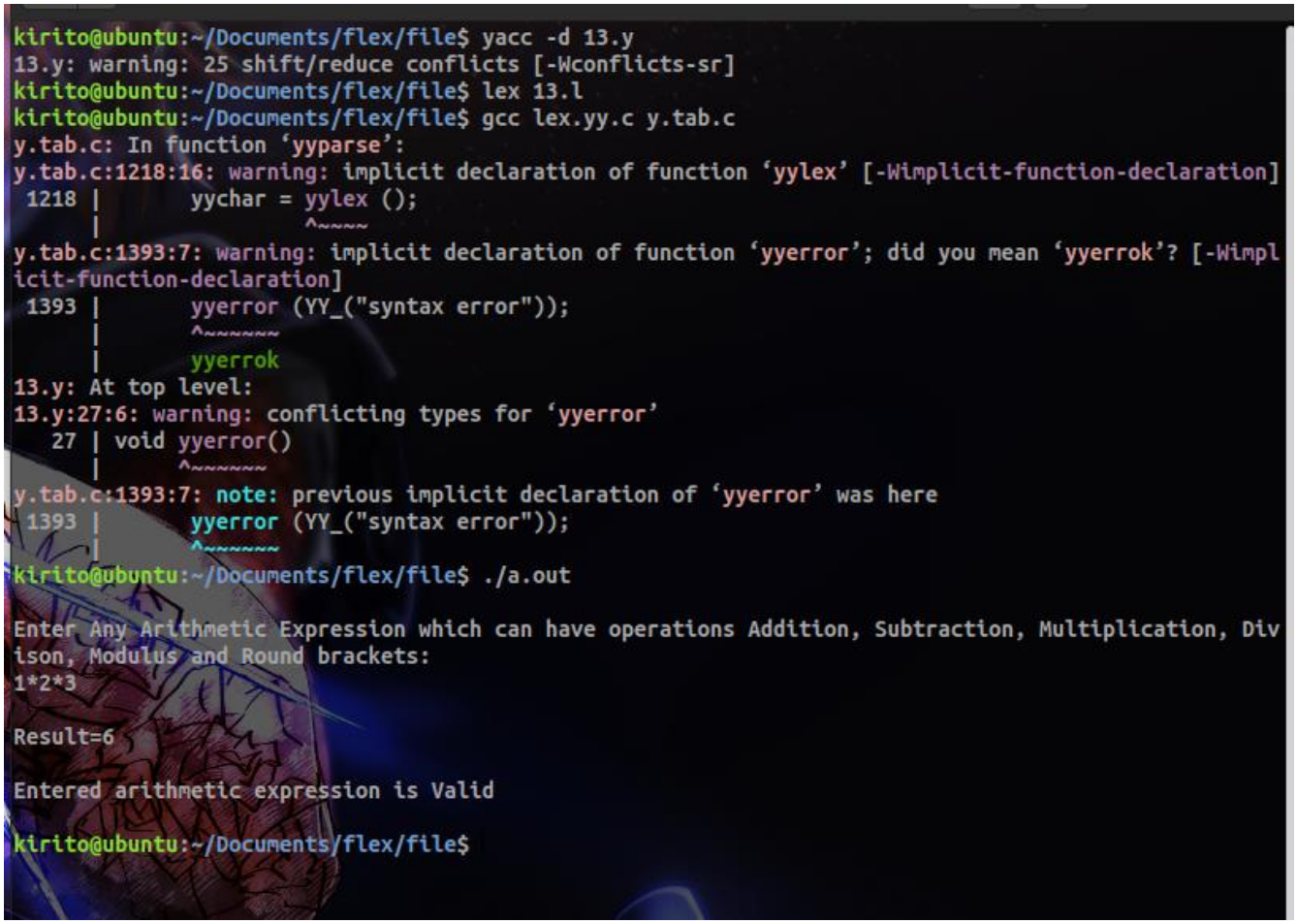
```
%{
#include<stdio.h>
#include "y.tab.h"
extern int yylval;
%}

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

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

Yacc Program:

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

%token NUMBER
%left '+' '-'
%left '*' '/' '%'
%left '(' ')'
%%

ArithmeticExpression: E{
printf("\nResult=%d\n",$$);
return 0;
};
E:E'+'E {$$=$1+$3;}
|E'-'E {$$=$1-$3;}
|E'*'E {$$=$1*$3;}
|E'/'E {$$=$1/$3;}
|E'%'E {$$=$1%$3;}
|'('E')' {$$=$2;}
```

```
| NUMBER {$$=$1;}
;
%%
void main()
{
   printf("\nEnter Any Arithmetic Expression which can have operations Addition, Subtraction,
Multiplication, Divison, Modulus and Round brackets:\n");
   yyparse();
   if(flag==0)
      printf("\nEntered arithmetic expression is Valid\n\n");
}
void yyerror()
{
   printf("\nEntered arithmetic expression is Invalid\n\n");
   flag=1;
}
```

Output:



2. Design YACC/LEX code to evaluate arithmetic expression involving operators +, -, * and / with operator precedence grammar.

Lex Program:

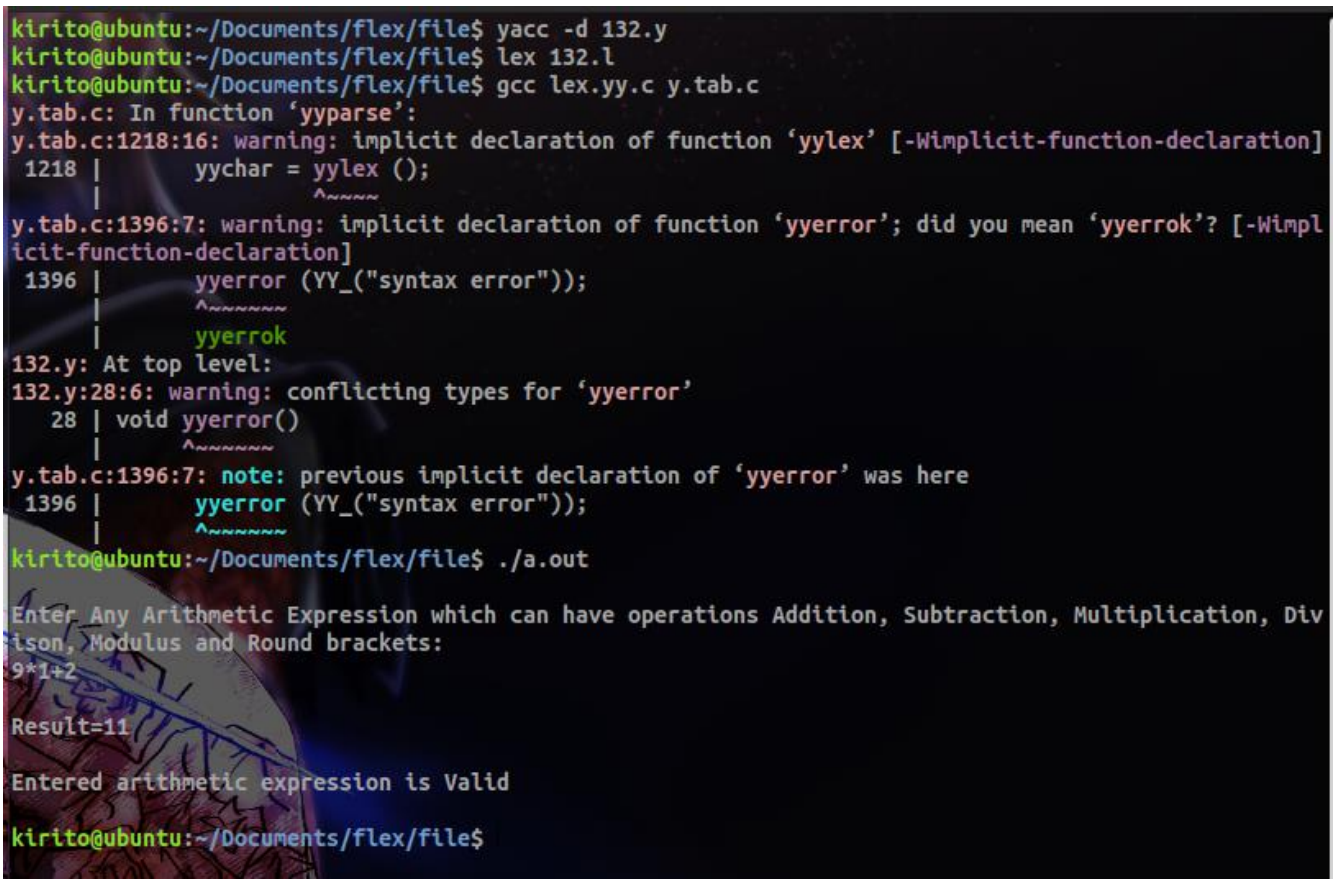
```
%{
        #include<stdio.h>
        #include "y.tab.h"
        extern int yylval;
%}

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

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

Yacc Program:

```
%{
#include<stdio.h>
int flag=0;%}
%token NUMBER
%left '+' '-'
%left '*' '/' '%'
%left '(' ')'
%%
ArithmeticExpression: E{
printf("\nResult=%d\n",$$);
return 0;
}
E:E'+'E {$$=$1+$3;}
|E'-'E {$$=$1-$3;}
|E'*'E {$$=$1*$3;}
|E'/'E {$$=$1/$3;}
|E'%'E {$$=$1%$3;}
|'('E')' {$$=$2;}
| NUMBER {$$=$1;}
;
%%
```

```
void main()
{
printf("\nEnter Any Arithmetic Expression which can have operations
Addition, Subtraction, Multiplication, Divison, Modulus and Round
brackets:\n");
yyparse();if(flag==0)
printf("\nEntered arithmetic expression is Valid\n\n");
}
void yyerror()
{
printf("\nEntered arithmetic expression is Invalid\n\n");
flag=1;
}
```

Output:



PROGRAM 14: Design YACC/LEX code to evaluate arithmetic expression involving operators +, -, * and / with operator precedence grammar.

Lex Program:

```
%{
   #include"y.tab.h"
   extern int yylval;
%}

%%
[0-9]+ {yylval=atoi(yytext); return NUM;}
\n return 0;
. return *yytext;
%%

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

Yacc Program:

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

%token NUM
%left '+' '-'
%left '*' '/'
%right NEGATIVE

%%
S: E {printf("\n");} ;E: E '+' E {printf("+");}
| E '*' E {printf("*");}
| E '-' E {printf("-");}
| E '/' E {printf("/");}
| '(' E ')'
| '-' E %prec NEGATIVE {printf("-");}
| NUM
{printf("%d", yylval);}
;
%%

int main(){
   yyparse();
}
int yyerror (char *msg) {
   return printf ("error YACC: %s\n", msg);
}
```

Output:

```
kirito@ubuntu:~/Documents/flex/file$ flex 14.l
kirito@ubuntu:~/Documents/flex/file$ yacc -d 14.y
kirito@ubuntu:~/Documents/flex/file$ cc lex.yy.c y.tab.c
y.tab.c: In function ‘yyparse’:
y.tab.c:1219:16: warning: implicit declaration of function ‘yylex’ [-Wimplicit-function-declaration]
 1219 |       yychar = yylex ();
      |                ^~~~~
y.tab.c:1388:7: warning: implicit declaration of function ‘yyerror’; did you mean ‘yyerrok’? [-Wimpl
icit-function-declaration]
 1388 |       yyerror (YY_("syntax error"));
      |       ^~~~~~~
      |       yyerrok
kirito@ubuntu:~/Documents/flex/file$ ./a.out

error YACC: syntax error
kirito@ubuntu:~/Documents/flex/file$ ./a.out
5+9*1-2
591*+2-
kirito@ubuntu:~/Documents/flex/file$ ./a.out
9+6-2
96+2-
kirito@ubuntu:~/Documents/flex/file$
```

PROGRAM-15: Design Desk Calculator using YACC/LEX code.

Lex Program:

```
%{
   #include<stdio.h>
   #include "y.tab.h"
   extern int yylval;
%}

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

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

Yacc Program:

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

%token NUMBER
%left '+' '-'
%left '*' '/' '%'
%left '(' ')'
%%

ArithmeticExpression: E{
printf("\nResult=%d\n",$$);
return 0;
};
E:E'+'E {$$=$1+$3;}
|E'-'E {$$=$1-$3;}
|E'*'E {$$=$1*$3;}
|E'/'E {$$=$1/$3;}
|E'%'E {$$=$1%$3;}
|'('E')' {$$=$2;}
| NUMBER {$$=$1;}
```

```
;
%%
void main()
{
   printf("\nEnter Any Arithmetic Expression which can have operations Addition, Subtraction,
Multiplication, Divison, Modulus and Round brackets:\n");
   yyparse();
   if(flag==0)
      printf("\nEntered arithmetic expression is Valid\n\n");
}
void yyerror()
{
   printf("\nEntered arithmetic expression is Invalid\n\n");
   flag=1;
}
```

Output: