1. The input to the program is a regular expression and a string. Output whether or not the string can be parsed from the regular expression.
    a. Computation of ϵ-NFA from regular expression.
    b. Simulation of ϵ-NFA on the given string.
2. The input to the program is a regular expression. Output the DFA with minimum states for the given regular expression.
    a. Conversion of regular expression to ϵ-NFA.
    b. Use subset construction to convert ϵ-NFA to DFA.
    c. Minimization of DFA.
3. The input to the program is a regular expression. Output the CFG which describes the language of the given regular expression.
    a. Conversion of regular expression to ϵ-NFA.
    b. Use subset construction to convert ϵ-NFA to DFA.
    c. Conversion of DFA to CFG.
4. The input to the program is a context-free grammar and a string. Outputs left most derivation of the string from the given grammar.
    a. Elimination of left recursion and performing left factoring for the grammar if it is not an LL (1) grammar.
    b. Computation of predictive parsing table.
    c. Simulation of predictive parser on the given string.
5. The input to the program is a context-free grammar and a string. Outputs reverse of the right most derivation of the string from the given grammar.
    a. Computation of LR (0) Sets of Items.
    b. Computation of SLR parsing table.
    c. Simulation of SLR parser on the given string.
6. The input to the program is a context-free grammar and a string. Outputs reverse of the right most derivation of the string from the given grammar.
    a. Computation of LR (1) Sets of Items.
    b. Computation of Canonical LR (1) parsing table.
    c. Simulation of Canonical LR parser on the given string.
7. The input to the program is a context-free grammar and a string. Outputs reverse of the right most derivation of the string from the given grammar.
    a. Computation of LALR parsing table.
    b. Simulation of LALR parser on the given string.
8. Develop a compiler for a subset of C language using tools Flex and Bison. The target environment will be SPIM simulator.
    a. Define a grammar for subset of C language.
    b. Using flex and bison build the compiler for that small language.
9. Develop a Web interface for a hybrid compiler similar to the one like www.ideone.com

10. Implementation of complete Front End of the compiler as given in Appendix A of Compilers: Principles, Techniques and Tools. The following modules are to be implemented
    a. Scanner
    b. Symbols
    c. Parser
    d. Inter
11. Using Flex and Bison tools, simulate a calculator that allows the following operations:
    a. Basic operators like +, -, *, /, %
    b. Unary operators like --, ++
    c. Parentheses
    d. Bitwise integer operations, AND (&), OR(|)

    Look at the table for sample input-output test cases

1.25 * 1.6	2.000000
1 * 2 + 3	5.000000
1 + 2 * 3	7.000000
1 + 2 * (3 + 2)	11.000000
1.5e1 – 0.8 * -2.5	17.000000
25 + 2 * -(3 * 5 % 4)	19.000000
---((3.4 + -(---4.3)))	-7.700000
15 – 1.5e1	0.000000
0.015e3 - -1500e-2	30.000000
1e7 * -3450000e-12	-34.500000
-0.00000345 / -1e-7	34.500000
1e1 * 10.000 * 1000e-2	1000.000000
0e-4	0.000000
0.0	0.000000
1e57 / 2e55	50.000000

12. Implementation of compiler for a programming language called “C--”. The following modules are to be implemented.
    a. Design and Implementation of Lexical analyzer for C--
    b. Design and Implementation of Syntax analyzer for C--
    c. Code Generation of basic functionality
    d. Complete C-- Compiler, Code, Project Report and Demo

C-- Grammar:

```
Program ------> VarDeclList FunDeclList

VarDeclList --> epsilon
               VarDecl  VarDeclList

VarDecl ------> Type id ;
                Type id [ num] ;

FunDeclList --> FunDecl
               FunDecl  FunDeclList
```

```
FunDecl ------> Type id ( ParamDecList ) Block

ParamDeclList --> epsilon
                  ParamDeclListTail

ParamDeclListTail -->  ParamDecl
                       ParamDecl,  ParamDeclListTail

ParamDecl ----> Type id
                Type id[]

Block --------> { VarDeclList StmtList }

Type ---------> int
                char

StmtList -----> Stmt
                Stmt  StmtList

Stmt ---------> ;
                Expr ;
                return Expr ;
                read id ;
                write Expr ;
                writeln ;
                break ;
                if ( Expr ) Stmt else Stmt
                while ( Expr ) Stmt
                Block

Expr ---------> Primary
                UnaryOp Expr
                Expr BinOp Expr
                id = Expr
                id [ Expr ] = Expr

Primary ------> id
                num
                ( Expr )
                id ( ExprList )
                id [ Expr ]

ExprList -----> epsilon
                ExprListTail

ExprListTail --> Expr
                 Expr , ExprListTail

UnaryOp ------> - | !

BinOp --------> + | - | * | / | == | != | < | <= | > | >=  | && | ||
```