References:

- Compilers : Principles,Techniques and Tools by Alfred V.Aho, Monica S. Lam, Ravi Sethi and Jeffrey D.Ulman .
- Modern Compiler Implementation in C by Andrew W.Appel
- Modern Compiler Implementation in JAVA by Andrew W.Appel
- lex & yacc, 2nd Edition by Doug Brown, John Levine, Tony Mason
- Flex & Bison by John Levine
- http://dinosaur.compilertools.net/
- https://docs.oracle.com/cd/E19504-01/802-5880/6i9k05dgg/index.html
- https://docs.oracle.com/cd/E19504-01/802-5880/6i9k05dgt/index.html
- https://www.ibm.com/docs/en/zos/2.4.0?topic=lex-input-language
- https://silcnitc.github.io/lex.html

Project

Design a programming language of your own

Tasks:

- **Create Compiler for your Language (You can use regional language as well)**
  - **Own Lexical Analyzer**
    - **Regular Expressions for your language**
    - **Handling of Errors**
    - **Actions**
  - **Tokens**
  - **Own Parser**
  - **Parse Tree**
  - **Intermediate Code Generation**
  - **Code Optimization**
  - **Target Code**
- **Write a sample source program for calculator in the language you developed**
- **Compile the program by your compiler**
- **Show that it is working correctly by producing outputs**

Assignment 1:

A. Develop the *components* of your own programming language.
B. Write regular expressions for each of them.
C. Draw the state transition diagram for each of the patterns and convert them together as an NFA.
D. Create a minimized DFA for the NFA.

E. Write Lex code implementing the patterns and corresponding actions.
F. Write codes for handling errors during lexical analysis.
G. Compile the Lex code and create your own lexical analyzer (L).
H. Write a small program in the language you have developed.
I. Compile and show that your lexical analyzer is creating correct tokens for it and handling errors correctly.

Assignment 2:

A. Create a parser for your programming language.
B. Show that this parser correctly parses the input token generated by your lexical analyser for any program written in your programming language as well as identifies error.
C. Write a simple program in your language with all kinds of tokens and keywords and show that your compiler is correctly detecting the tokens and errors. Parse the program using your parser. Print step by step parsing process.

Assignment 3:

A) Generate syntax directed translations for your grammar. You may need to modify your grammar to apply the semantic rules.
B) Apply the concept of backpatching to give destination address of jump statements correctly.
C) Show that for any types of arbitrary blocks of codes written in your programming language, your compiler is able to generate the appropriate three address codes.
D) Show the derivation of these inputs from your grammar. Draw the parse trees with the semantic translations and show the parsing steps which generates the correct three address codes through backpatching.

Assignment 4:

A) Apply optimization techniques over your intermediate code generated in the front end pf your compiler.
B) For the sample codes written in your language (in previous assignments), show that your compiler is able to identify the basic blocks and apply optimization techniques to derive optimized intermediate code.
C) Show the flow graph and DAG for the sample codes of Q. 4B. Draw all steps.

Advanced (optional): Derive the target code for your compiler.