1.) For the grammar from homework 1, rewrite it to encode precedence and left factor. Not (!) should be highest precedence, ^ middle precedence, and v should be lowest.

\[
\begin{align*}
E &\rightarrow E \wedge E & E \rightarrow \text{true} \\
E &\rightarrow E \vee E & E \rightarrow \text{false} \\
E &\rightarrow \neg E
\end{align*}
\]

2.) Construct an LL(1) parse table for your grammar from 1. Show the First and Follow sets you generated.

3.) Show the LL(1) parsing action list for the input:

\[!\text{false} \vee \text{false} \wedge \text{false}\]

4.) Based upon your grammar for 1), write a recursive descent parser. On the next page, I have provided a skeleton and minimal test suite for you to follow. You do not need to support whitespace and should use T for true and F for false. Fill in the skeleton with the implementation of a function per nonterminal in your grammar. Print out your code (it shouldn’t be more than 2 pages) and submit it.
class RecursiveDescent {
    public static void main(String[] args) {
        String[] tests = {
            "T", //Accept
            "!F", //Accept
            "T^F", //Accept
            "T^T^", //Reject
            "!TvF", //Accept
            "!vفذ", //Reject
            "!F", //Accept
            "!T^!T" //Accept
        };

        //Note: We are not writing an interpreter, only accepting valid strings
        //and rejecting invalid ones. That is, I don’t care whether the answer
        //is true or false for the valid ones.

        for (String test: tests) {
            if (S(new StringBuffer(test))) {
                System.out.println(test + " : Accept");
            } else {
                System.out.println(test + " : Reject");
            }
        }
    }
}

/**
 * Assuming your start symbol is S
 * @param str - the StringBuilder of your input (mutable to “eat” tokens)
 * @return whether the parse is valid at this point or not
 */
private static boolean S(StringBuilder str) {
    //Use str.charAt(0) to peek at the first character
    //Use str.deleteCharAt(0) to eat the first character

    //based upon the character (or not), recursively call
    //other production functions

    //Return false if you encounter something bad or return the
    //return values of the recursive calls you make
}

//Add your additional functions here.